# ENERGY-EFFICIENT Q-LEARNING FOR COLLISION AVOIDANCE OF AUTONOMOUS ROBOTS

A Thesis

by

SEUNGJAI AHN

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

| | |
|---|---|
| Chair of Committee, | Peng Li |
| Committee Members, | Weiping Shi |
| | Eun Jung (EJ) Kim |
| Head of Department, | Miroslav M. Begovic |

May 2017

Major Subject: Computer Engineering

# ABSTRACT

Recently, many companies have been studying intelligent cars, and improvements in sensor technology and computing are required. The intelligent cars use GPS to know where they are. The cars use sensors to detect objects in all directions, including people, vehicles and animals. Based on what the sensors detect, the software processes the information to help the cars predict what all the objects around the cars might do. Based on the prediction, the cars choose a safe speed and trajectory for themselves. Therefore, the current research on intelligent cars is mainly focused on collision avoidance because of safety. Every year, there are over two million traffic-related deaths worldwide. This number could be dramatically reduced, especially since 94% of accidents in the U.S. involve human error.

In this work, we show the design, development, and testing of an autonomous ground vehicle for testing energy-efficient of Q-learning in robotics. The vehicle platform is based on the Boe-Bot chassis, which is an education shield for Arduino. The Boe-Bot is a common research testbed that encourages the use of low-cost hardware and open-source software. The research platform uses Raspberry pi 2 Model B as its on-board computer for handling Q-Table collision avoidance like processing sensor data and computing deciding actions. The robot is then tested by using a modified Q-Learning algorithm to avoid collisions. We design hardware-friendly software, and the robot can use the same Q-Table as the software. The robot runs in several maps to find relationships between reward function and time and energy consumption. One target of this work is to find a relationship between reward function and time and energy consumption. This study will help us to understand feasibility of saving time and energy for intelligent cars.

The aim of this master's thesis is to investigate energy-efficient Q-learning collision

avoidance in robotics. Unlike, the existing research, we consider saving energy for collision avoidance. Furthermore, we could use same the Q-Table on the actual robot because we designed the software as hardware-friendly.

# DEDICATION

To my father, my mother, and my sister.

# ACKNOWLEDGMENTS

Thanks to Dr. Li, Myunseok Shim and Paul Crouther for their contributions and help in this research.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

# LIST OF TABLES

TABLE Page

# 1. INTRODUCTION

## 1.1 Motivation

Recently, machine learning has been programmed to optimize a performance criterion using example data or previous observations[5][6]. However, robots behaving in intelligent ways, with respect to their surrounding environments to avoid obstacles will need to have some methods of in-built learning mechanisms. One of the most popular machine learning methods is Q-learning. Q-learning has been widely used for autonomous driving to avoid collision because Q-learning is a simple algorithm and chooses optimal action-value function. As is commonly known, implementation on a real hardware poses some troubles. Many companies work on autonomous driving applications. They have almost finished designing and building the autonomous driving car, but they are still considering and testing safety. The collision avoidance is the most important role in autonomous driving cars for safety. For the important role, Google and Tesla are using many kinds of sensors for safety. For example, lasers, radars, and cameras detect obstacles in all direction.

It is extremely hard for robots to learn human skills. Understanding the robots'surrounding environments, localizing and safely navigating through an environment are examples of tasks that are difficult for the robots. For example, distinguishing opening doors and windows could be difficult to the robot.

This thesis examines not only the Q-learning algorithm but also energy-efficient Q-learning. The energy-efficient Q-learning is modified via adjusting reward function. The two Q-learning methods were investigated on python and pygame environments. Additionally, the testbed was designed for these two algorithms, and the two Q-learning algorithms were tested on the testbed.

## 1.2 Literature Review

For autonomous driving, the ability to avoid obstacles in the environment is the most important required capability. Obstacle avoidance is the ability to be safe during driving. In order to avoid obstacles, many kinds of sensors are used for safety. For example, lasers, radars, and cameras detect obstacles in all directions. Robot Collision Avoidance denotes the robot's skill to know distances between the robot and any objects near the robot. The distance information does not have to be exact metric. That's because humans cannot measure the distance precisely. For implementation on a real robot, the Q-learning algorithm popular reinforcement learning method because Q-learning is off-policy and simple algorithm.

Q-Learning has been used in several real robot hardware experiments on a wide variety of platforms, including Lego NXT Robot [2], Pekee[12], and Amigo Robot[1].

As is commonly reported, implementation on real hardware robot shows several challenges. In simulation, the problem can be solved with Q-learning; however, the required learning time can exceed the mean time in real world. For this problem, the robot moves around to learn on the real world, so the current researchers improve Q-Learning for faster convergence.

Lazhar Khriji and Farid Touati used a Pekee robot for mobile robot navigation via Q-learning similar to the ideas discussed in this thesis. The QLearning algorithm is applied to coordinate between these behaviors, which allows for a great reduction in learning convergence times. This paper showed Q-learning can be used as a successful method for robot collision avoidance. Therefore, simulation and experimental results confirmed that the robot could learn and choose the best action in various environments. It means that the robot learned the desired results correctly.[7]

In another paper, Lvwen Huang, Dongjian built their own indoor robot. The robot has

ultrasonics sensors and infrared sensors to detect obstacles around the robot. Unlike other experiments, the robot used optimal path and Q-value table being found and obtained from PC navigation simulation. After gaining the table, the robot moved in indoor environments using Q-learning. In summary, the study used a Q-Learning based navigation method to solve the moving control along a specified path of real indoor mobile robot.[8]

# 2. REINFORCEMENT LEARNING

## 2.1 Machine Learning

Machine learning is a computer learning without being explicitly programmed. The field of machine learning is broken up into three categories: supervised, unsupervised, and reinforcement learning.

In supervised learning[9], after an action is taken, a learner tells an agent what the best action would have been. In supervised learning, the agent has a teacher, and the teacher has knowledge. The agent tries to emulate the teacher gradually. In unsupervised learning[9], the learner does not tell the agent afterwards what the best action would have been. Therefore, the agent learns based on task-independent measurements of the quality of representation. In reinforcement learning[9], when the agent takes the action, the agent learns how to achieve a task in the environment by trial-and-error interactions.

## 2.2 Reinforcement Learning

The reinforcement learning problem is posed as follows: given a discrete set of states, a discrete set of actions, a policy, and reward function, find the optimal policy that maximizes the future reward. A policy defines a learning agent's way of performing at a given state. Simply, it is just a plan, and it is a mapping from some states of the environment for an action to be taken when in those states.[1]

For example, when the agent takes an action, the environment produces a new state and reward. According to the reinforcement learning algorithm, the agent updates its policy from the current state, next state and reward. In Figure 2.1,there is a drawing illustrating the basic reinforcement learning model.

The best method to imitate learning behavior in humans is reinforcement learning. In reinforcement learning, the learner is a decision-making agent that takes actions in an en-

vironment and receive a reward for the actions in trying to solve a problem. After finishing trial-and error, it should learn the best policy to maximize the future total reward.

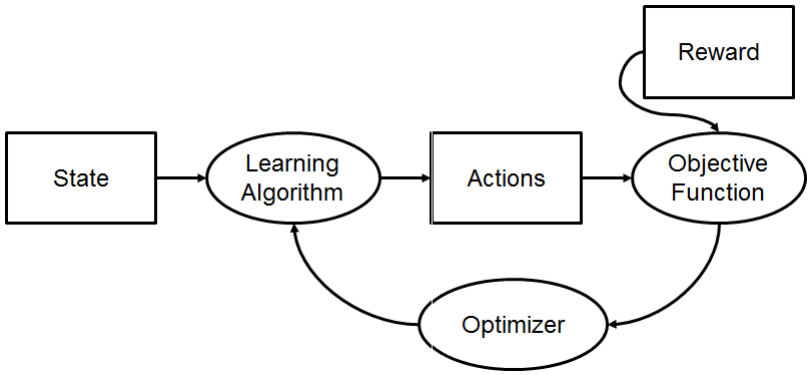


Figure 2.1: Reinforcement learning flowchart[1]

## 2.3 Markov Decision Process (MDP)

A reinforcement learning task that satisfies the Markov property is called a Markov decision process, or MDP[1]. A Markov decision process consists of:

- An agent
- An environment
- A finite set of states
- A finite set of actions
- A reinforcement reward function
- A state transition function

- Discount factor
- Learning rate

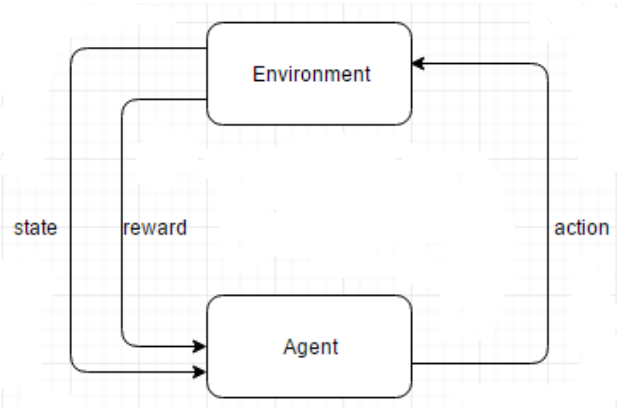


Figure 2.2: Markov decision process description picture[1]

When an agent interacts with an environment, the environment always provides states that the agent observes. The agent takes actions in the environment which provide a reward. A state transition function is a plan of how the agent decides which actions to take given the state. There are two factor to consider: discount factor and learning rate. Discount factor describes how much of a future reward the agent wants. Learning rate means the agent considers how fast the agent wants to change the state transition function. If the learning rate is too large, the agent will continue to stay within local optima or around the place of the agent's converge.

Markov Decision Process is particularly important to the reinforcement learning. A necessary condition for a reinforcement learning problem to be modeled as a Markov Decision Process is that the state transitions are independent of any previous environment states and actions. In other words, the state transitions have to be independent of the actions of the agent until this point, and only depend on the current state.

TD(Temporal-Difference) methods can learn directly from experience in an environment and update estimates based on other learned estimates before a final outcome. TD methods use experience to solve problems because robots learn their estimates in part of other estimates. There are two main classes: on-policy and off-policy. The two classes are presented in next sections.

### 2.3.1 SARSA(On-policy)

SARSA(State-Action-Reward-State-Action) is one type of on-policy reinforcement learning that estimates the value of a policy being followed. A SARSA consists of:

- An agent
- An environment
- A finite set of states
- A finite set of actions
- A reinforcement reward function
- A state transition function
- A q-value
- Discount factor
- Learning rate

An agent will interact with the environment and update a Q-value based on actions taken. Q-value means the agent's current estimate reward in an state with an action. An experience in SARSA means that the agent was in a state, did an action, received a reward, and arrived at another state, from which it decided to do another action. Based on this, the new experience is updated to the Q-value.

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize $s$
  Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
  Repeat (for each step of episode):
    Take action $a$, observe $r$, $s'$
    Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma Q(s', a') - Q(s, a)\big]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until $s$ is terminal

Figure 2.3: One sample SARSA code[1]

### 2.3.2 Q-Learning(Off-policy)

One of the simplest methods to implement is Q-Learning, and it is also a widely used model-free reinforcement learning and proposed by Watkins[1]. A Q-Learning consists of:

- An agent
- An environment
- A finite set of states
- A finite set of actions
- A reinforcement reward function
- A state transition function
- A q-value
- Discount factor
- Learning rate

Q-Learning has the Q-value for predicted future rewards, and it is stored in a table, which is a lookup table. If the discount factor is zero, it makes the agent myopic, which is when the agent only cares about current rewards. However, if discount factor is close to one, it will make the agent try for a long-term high rewards. For learning rate, if it is zero, the agent does not learn anything. If learning rate is close to one, the agent only cares about the most recent rewards.[10]

Q-Learning is similar as SARSA, but there is one difference. During the update, Q-learning uses the maximum reward of available actions instead of Q-value based on actions take. Q-Learning needs to explore all of the possible states to converge correctly. There are several methods to solve this problem. Simply, an agent always chooses an action for given state with the highest Q value. If there is a tie, the agent choose an action at random. However, it is extremely inefficient, and the agent might miss a lot of possible states. To improve efficiency, epsilon term can be used to solve the exploration problem. Briefly, if a random number is less than epsilon value, the agent does not follow an action with the highest Q-value. The agent choose an action at random. Detailed explanation for the epsilon greedy method in next section.[2][11]

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize $s$
  Repeat (for each step of episode):
    Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Take action $a$, observe $r$, $s'$
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$;
  until $s$ is terminal

Figure 2.4: One sample Q-learning code[1]

### 2.3.3 Epsilon-greedy Method

Epsilon-greedy method is used for explorative policy, which chooses random actions with a fixed probability and the action with highest Q value otherwise. Learning rate controls how fast the policy is modified. One expects to start with a high learning rate, which allows fast changes, and low the learning rate as time progresses. Discount factor determines the importance of future rewards. If discount factor is zero, the agent only considers current reward, and if discount factor approaches one, the agent cares about the future rewards. This method is called off-policy because the Q value of the best next action is used without using the policy that can decide an action.

$$\text{Action} = \begin{cases} \text{random action from finite set of actions, if probability} < \epsilon \\ \text{selecting one of the learned optimal actions, otherwise} \end{cases}$$

Figure 2.5: Epsilon-greedy method

### 2.3.4 Q-Learning Flowchart

In this section, Q-Learning algorithm is described step by step. First, an agent initializes a Q-Table, and receives the first state from an environment, and decides an action based on the state or randomly decides an action. After the agent finishes performing the action, the agent updates Q-value via the update function. Again the agent receives another state from the environment. The agent keep repeating the processing until the number of repetitions is met.

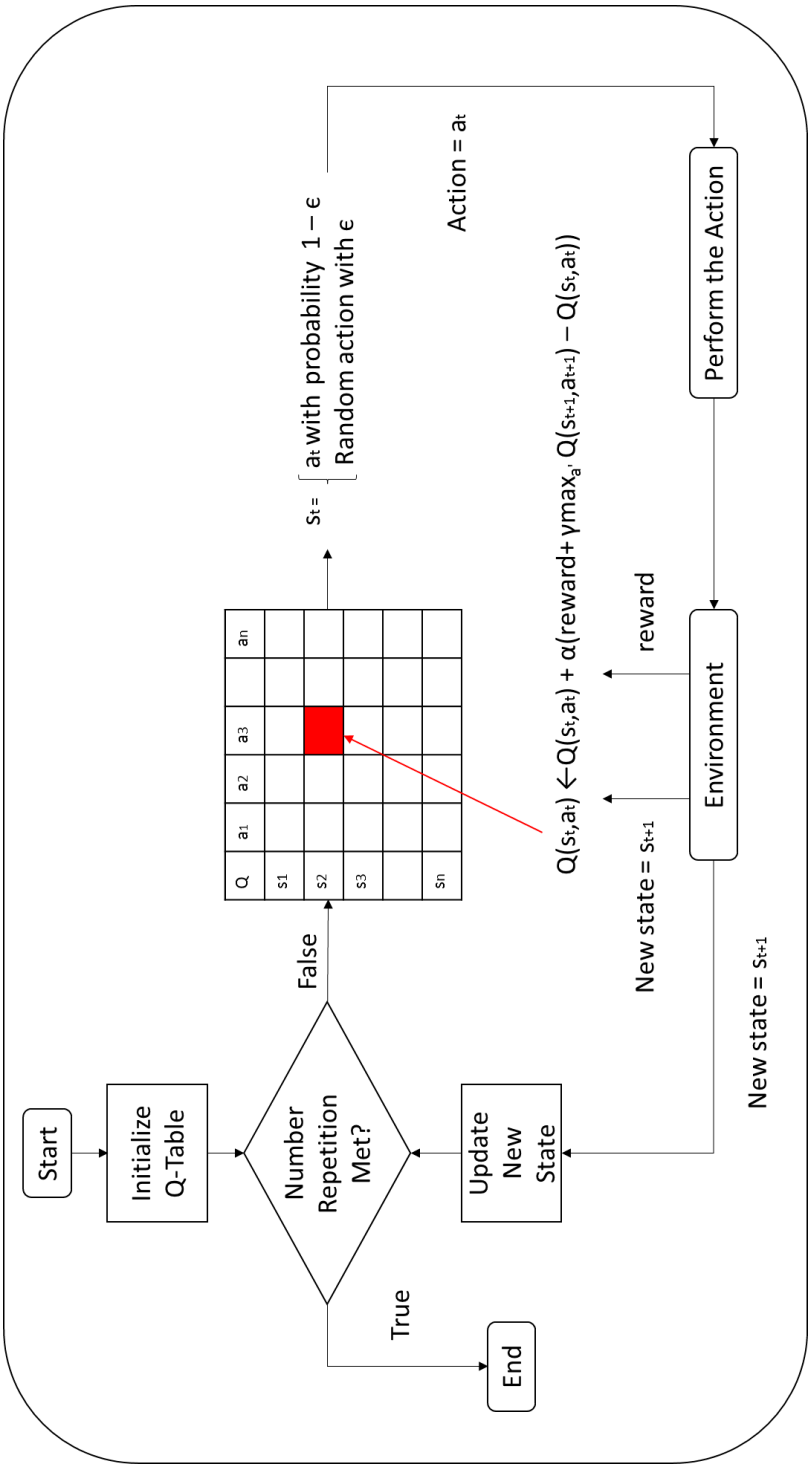


Figure 2.6: Q-learning flowchart[2]

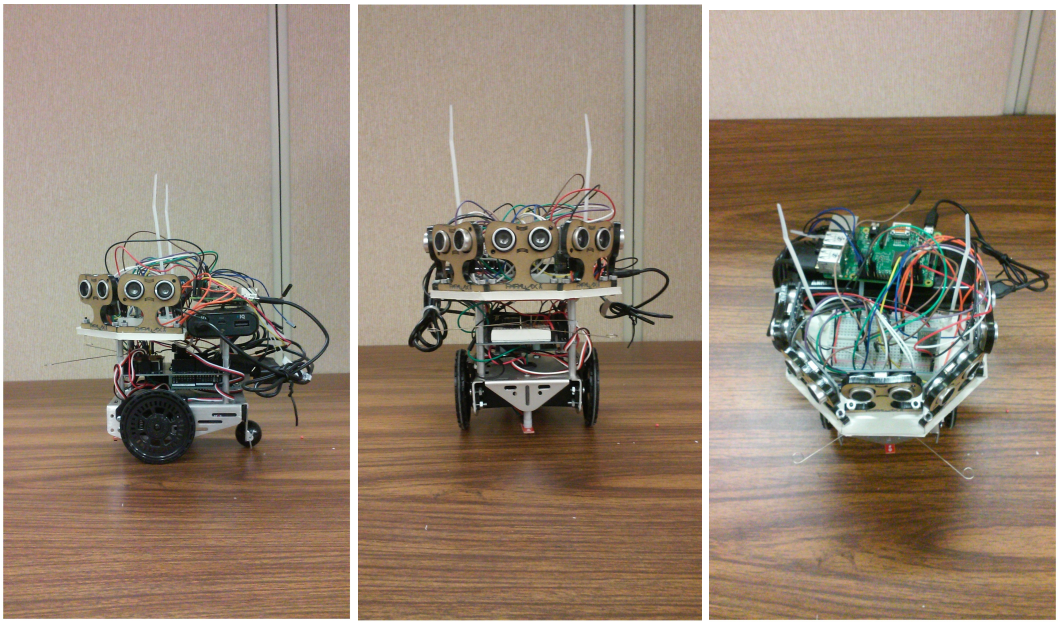

Figure 3.1: TestBed pictures

## 3.1 System Overview

In this experiment, a Boe-Bot shield with arduino uno is used. The robot has raspberry Pi 2 for storing a Q-table. The robot is equipped with 5 ultrasonic sensors, but only three of them are used for now. If we use all five ultrasonic sensors, the robot needs more time to converge, so we used only three of them to save time. A line follower sensor is attached below Boe-Bot for detecting black line. raspberry pi and arduino communicate with each other via I2C communication. One external portable battery is connected to raspberry pi 2 for power.

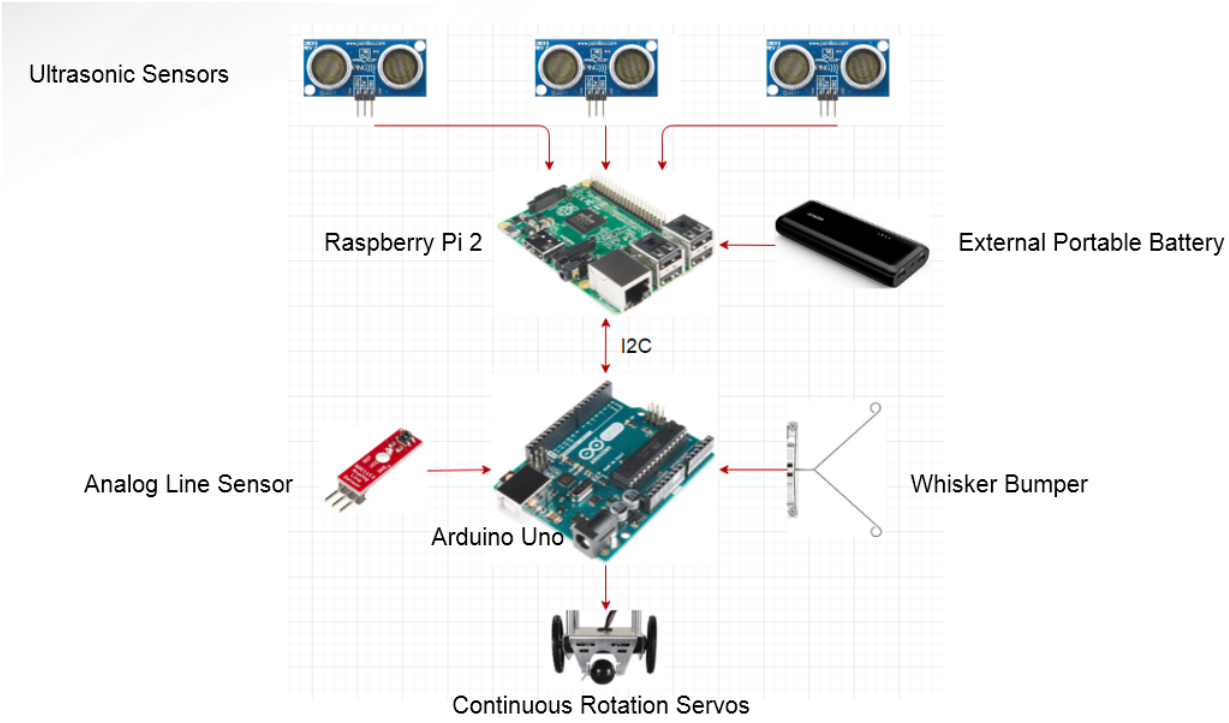


Figure 3.2: Overall architecture of the robot

## 3.2 Part Lists

Bot-Bot, arduino uno and a top metal plate are used building for a platform because Boe-Bot works well with arduino uno, and a top metal plate is strong and light. Also, the chosen chassis for Boe-Bot is high-quality aluminium. Raspberry pi 2 can create and store Q-Table. Five ultrasonic sensors are attached on the Boe-Bot. For detecting a destination, a line follower sensor is attached under the Boe-Bot.

### 3.2.1 Boe-Bot

The Boe-Bot has two continuous rotation servos. Moreover, the Boe-Bot shield is compatible with an arduino uno, so it makes easy to build circuits and connect servos to the arduino uno. A solderless breadboard, servo ports, and the Shield makes arduino uno programming easy.

### 3.2.2 Arduino Uno

Arduino Uno is a microcontroller board based on the ATmega328P, and allows for faster transfer rates and more memory. It has fourteen digital I/O pins and six analog pins. Moreover it has 32KB of flash memory amd 2KB of RAM, and it can be simply connected to a raspberry pi 2 with a USB cable. It is the most famous microcontroller and it is easy to use for controlling sensors and servos. It is also an open source physical computing platform. Lastly, arduino uno does not need operating system overhead. Therefore, arduino uno is microcontroller for the Boe-Bot.

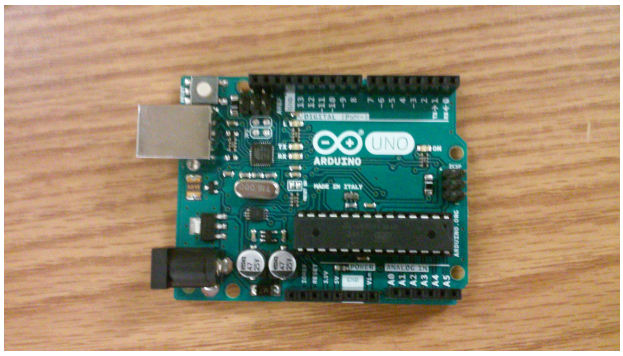


Figure 3.3: Arduino uno picture

### 3.2.3 Top Red Metal Plate

The top metal plate is fabricated to attach ultrasonic sensors and Raspberry pi 2 to an Boe-Bot. This plate is a milled aluminum plate that is strong enough to secure the sensors and Raspberry pi 2. The plate dimensions are 70mm x 80mm x 1mm and weight is 10.3g.

### 3.2.4 Raspberry Pi 2

Raspberry pi 2 is similar to a fully functional computer, so it can run operating system(OS) like linux, windows, and mac. Raspberry pi 2 has an ARMv7 Quad Core Proces-

sor, running at 900MHz, and it can run an operating system from a microSD card. It has four built-in USB ports and a HDMI port. A keyboard and a mouse are connected to it via USB ports, and a monitor is connected to it via a HDMI cable. Using these devices, python codes can be programmed. For power supply, an external battery should be connected to the micro-USB port. It has forty GPIO pins: GPIO, UART, I2C, SPI as well as 3.3 and 5V sources. Raspberry pi 2 is attached to the top metal plate. Raspberry pi 2 can simulate python codes and store Q-tables. Therefore, the robot does not have to update Q-Table during running. It uses the given Q-table.

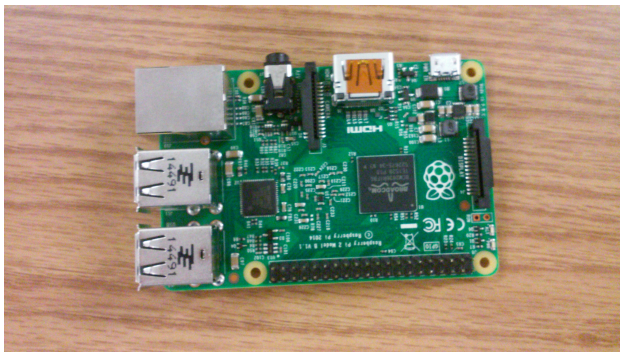

Figure 3.4: Raspberry pi 2 model B+

### 3.2.5 External Portable Battery

For the thesis, Anker 2nd-generation astro E5 is used for power supply. This battery has high-capacity 16750mAh, and when it connects to the testbed, it delivers its fastest possible charge up to 3A. Therefore, the testbed can run for the thesis. Additionally, there are four LEDs on the battery, and the LEDs show how much battery life remains.


Figure 3.5: External portable battery

### 3.2.6 Ultrasonic Sensor

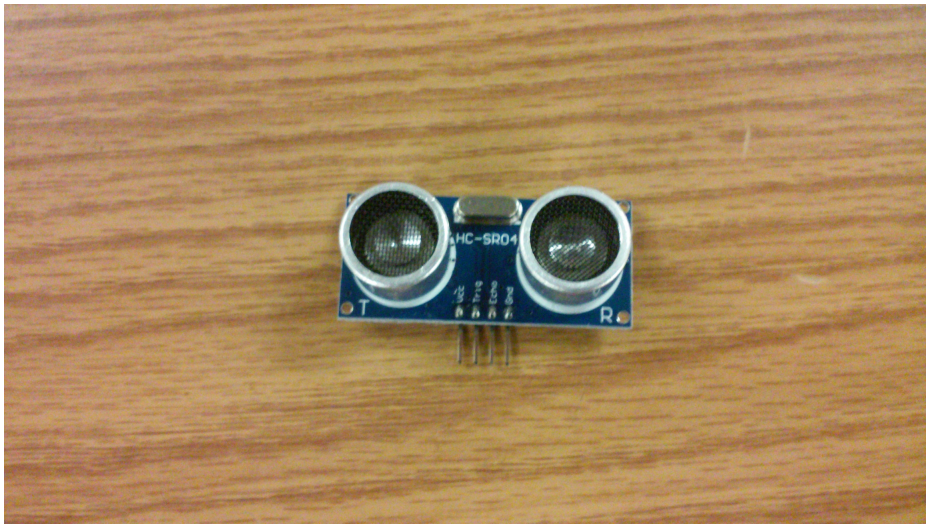


Figure 3.6: Ultrasonic distance sensor

The ultrasonic ranging sensors include an ultrasonic transmitter and a receiver and con-

trol circuit. It has four pins: VDD(Power), Trig(Trigger), Echo(Receive), and GND(Ground). The testbed can supply a short pulse to the trigger input to start measuring a distance, and the sensor detects its echo. After detecting the echo, the testbed can calculate the distance through a time interval between sending a trigger signal and detecting an echo signal.

There five ultrasonic sensors are attached on the Boe-Bot, but only three ultrasonic sensors are used for this project. Each ultrasonic sensor returns distance measurements to Raspberry pi 2. These values give measures of distance to obstacles in the proximity of the sensors.

- Limitations
  - Obstacles cannot be soft objects.
  - Ultrasonic sensor cannot detect longer than 300 feets.
  - At least part of the object in order to provide a surface capable of reflecting the ping wave.
- Solutions
  - No soft objects.
  - The robot will perform indoors.
  - Rough surface obstacles will be used.

For using rough surface for obstacles, single sided corrugated card board is attached on the walls to create rough surfaces for obstacle. . With using the cardboard, an echo signal can be possible to back to an ultrasonic sensor, so the testbed can measure a distance.

Figure 3.7: Solution for reflection problem

### 3.2.7 Digital Encoder for Boe-Bot

A digital encoder is known as wheel odometry. The digital encoder provides wheel rotation feedback that the Boe-Bot can use to perform more consistent maneuvers than not using a digital encoder. Also, this device is simple and easy to install and did not rely on a separate co-processor to mange the encoder pulses. This device mounts next to the wheels emitting infrared light and detecting reflections as the spokes pass by. The wheels have evenly spaced holes - enough for thirty two pulse edges per revolution. By incorporating the encoders, the robot can tell how far each wheel has turned and is able to read the wheels' movements. Based on what the robot reads about the wheels' movements, it can figure out a desired destination. Therefore, the robot can move and rotate more stably and precisely, using this device.

### 3.2.8 Line Follower Sensor

A line follower sensor is attached below the Boe-Bot. The sensor has a 3 pins: VDD, GND, and OUT. It detects reflected light coming from its own infrared LED, and it measures the amount of the reflected infrared light. Therefore, it can detect black color so that the robot can detect destination.

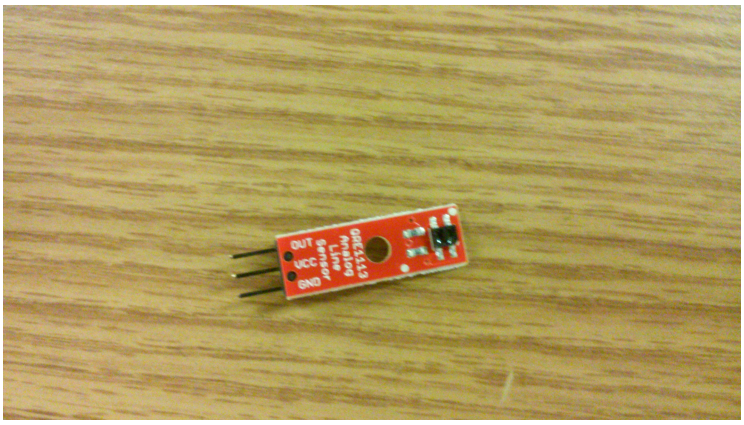


Figure 3.8: Line follower sensor

### 3.2.9 Whisker Bumper

The whisker bumper is attached to the front of the testbed, like a cat's whiskers. It can provide inputs that trigger some form of programmed output. The testbed can stop moving if the whisker bumper is pressed. The whisker circuit is wired for active-low output, so they each send a low signal when they are pressed and a high signal when they are not pressed. Whenever the whiskers are pressed, and arduino uno gets zero value, the testbed stops at once. Therefore, the whiskers can protect the testbed from collision.

### 3.3 Extra Set-up

For the robot, There are three set-ups needed for performing collision avoidance. One is I2C communication between raspberry pi and arduino uno. The second set-up is WI-FI for autonomous accessing, so any laptop or desktop can access to the raspberry pi. Finally, Arduino-IDE should be installed on raspbian.

#### 3.3.1 Install Arduino-IDE

Regarding recent raspbian, there is no Arduino-IDE software, Simply, 'sudo apt-get install arduino'command can install Arduino IDE on raspbian. After installing the software, arduino uno codes can be uploaded from raspberry pi 2.

#### 3.3.2 I2C Communication

There are several methods to communicate between Raspberry pi 2 and Arduino Uno. One of methods is communication via USB cable. This method does not need extra wiring to communicate between raspberry pi 2 and arduino uno. Also, the method does not need any instalment on raspberry pi 2. However, during the testing, some of character were missing with no reason. Arduino Uno is connected USB cable to Raspberry pi 2, so USB cable cannot handle Power supply line and communication. Another method involves using a bluetooth device, but it needs more wiring than I2C communication. Therefore, I2C Communication is the best communication for the robot. For wiring I2C between Raspberry pi and Arduino Uno, there are three connections: ground connection, SDA connection and SCL connection. For I2C communication, Raspberry pi 2 and Arduino Uno have to be connected like below Figure 3.9.

After wiring, several files need to be modified to run I2C communication on raspberry pi 2. On /etc/modprobe.d/raspi-balcklist.conf, blacklist i2c-bcm2780 line should be removed. On /etc/moduels, i2c-dev should be added. To install I2C tools, command line

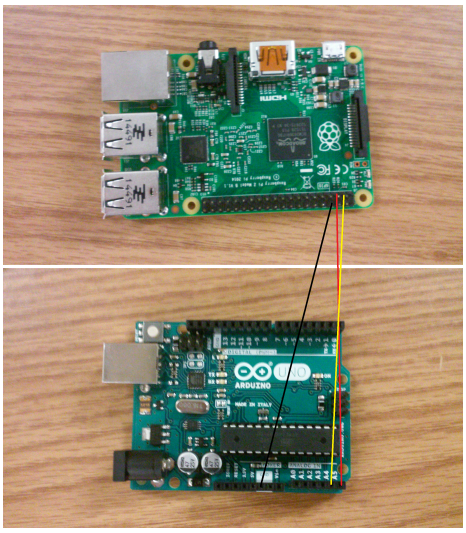

Figure 3.9: I2C communication wiring between raspberry pi 2 andvarudino uno (ground connection - black wire, SDA connection - yellow wire, SCL connection - red wire)[3]

'sudo apt-get install i2c-tools'is required. For using I2C, python-smbus is required. This command 'sudo apt-get install python-smbus'provides I2C support for the python.

### 3.3.3 WI-FI on Texas A&M University

WI-FI is necessary for controlling the testbed. A WIFI USB Adapter is attached to Raspberry Pi. However, raspberry pi cannot connect to university wifi itself, so for using university WIFI, two files should be modified before running the testbed. The two files are $/etc/network/interface$ and $/etc/wpa_supplicant/wpa_supplicant.conf$. The two files should be same as below.

After this step, Raspberry Pi can access University WIFI. With WIFI, PC or Laptop can access to Raspberry pi 2 to control it.

```
auto lo
iface lo inet loopback
auto eth0
allow-hotplug eth0
iface eth0 inet manual
allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

Figure 3.10: Content of /etc/network/interface[4]

```
network={
scan_ssid=1
ssid="tamulink-wpa"
key_mgmt=IEEE8021X
auth_alg=OPEN
eap=PEAP
identity="Your TAMU Net ID"
password="PASSWORD"
phase2="auth=MSCHAPV2"
}
```

Figure 3.11: The lines should be added on $wpa_supplicant.conf$ [4]

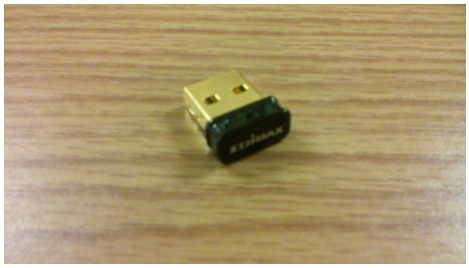

Figure 3.12: Wifi usb adapter

### 3.4 Problem for Using Other Sensors

There are other sensors can be used for the thesis. In other research, people are using location of an agent in simulation. An IR-Beacon Sensor can be used for direction of an

agent in other research. 360 degree laser sensor for distance measurement is a famous device for collision avoidance research. However, these devices have several problems for the thesis.

### 3.4.1 Indoor GPS

Many researchers are working on indoor GPS devices. There are several methods to localize an agent. One of the methods is using Blue-tooth. Using distances between an agent and base stations, the agent can assume the location. However, the device is not utilized for collision avoidance. First of all, measuring distance via bluetooth can involve noise if there are some obstacles near the agent. Therefore, indoor gps cannot be used for the thesis.

### 3.4.2 IR Beacon

IR-Beacon has a reflection problem. Because of reflection, ir-beacon can detect wrong direction of an agent, causing the agent to decide a wrong action.

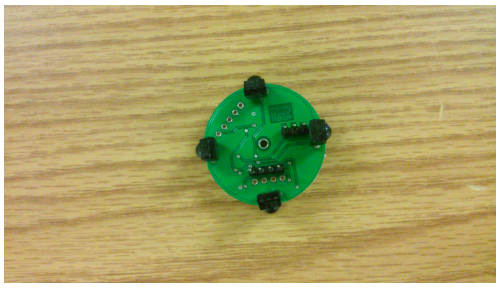

Figure 3.13: Ir-beacon device

### 3.4.3 360 Degree Laser Sensor for Distance Measurement

Some researchers are using a 360 degree laser scanner device. However, the device is fragile and expensive for the thesis.

# 4. METHODOLOGY

## 4.1 The Task and Environment

The task for the robot is to learn obstacle avoidance and arrive at a specific destination as shown in other papers[12][13][14]. The testbed robot has three ultrasonic sensors to measure distances between the robot and obstacles. To detect a destination, a line follower sensor is attached under the robot. When the sensor detects a black tape, the robot will be stopped by itself. The robot can perform three different movements: forward, left, and right. Before the robot arrives at the destination, the robot uses data from three ultrasonic sensors. Based on the data, the robot chooses one movement. Additionally, the robot should consume less energy. The energy-efficient Q-learning algorithm was tested on both the computer and the robot. To save time, the robot used the Q-Table from the software's Q-Table, so the robot did not have to learn anything during the demonstration.

## 4.2 Transfer Q-Table from Software to Hardware

Using cPickel function, a Q-Table from the software simulation can be stored on a PC, but when the Q-Table was transferred from PC to raspberry pi, an error occurred. That's because raspberry pi and Windows have different versions of skeletons. Because of the difference, a Q-Table should be created on raspberry pi 2. For using a Q-Table from the software simulation, python codes should be simulated on raspberry pi which is mounted on the testbed.

## 4.3 Software Set-up, States, and Actions

Python 2.7 and pygame[15] are used for simulation environments, and only python standard libraries are utilized. Each line detects distances between the robot and obstacles. An ultrasonic sensor is best at a 30 degree angle, and it detects the closest object. Angle

between two lines is 15 degrees, so three lines are equal to one ultrasonic sensor. If each line measures different distances, the agent choose the shortest one.

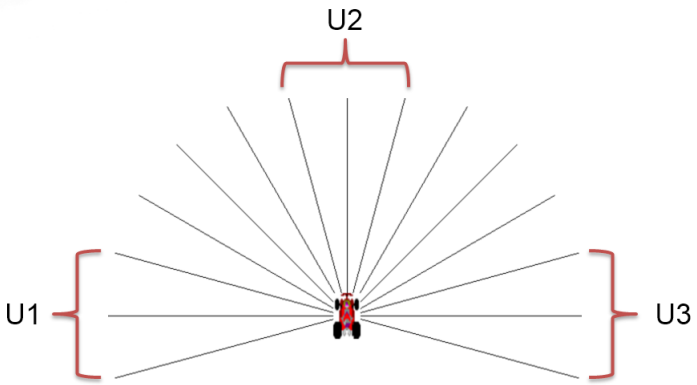


Figure 4.1: Picture of the agent on pygame and python

### 4.3.1 States Sensor-Space Division

Distance thresholds are defined to discrete states in Table 4.1. There are three ultrasonic sensors, to the total of states is 64.

Table 4.1: Discreted distance

| Distance | Value Given by the Sensor (pixels) |
|---|---|
| Too Long Range | Longer than 300 |
| Long-Range | 200 to 300 |
| Mid-Range | 100 to 200 |
| Short-Range | 0 to 100 |

### 4.3.2 Action-Set Construction Forward Left Right

The actions are defined by three discrete robot actions. The robot has three basic actions: move forward (F), turn left and move forward, and turn right and move forward. Only three actions are used for the study because more actions need and more time to converge. Forward action means the agent moves forward 20 pixels. Left actions means agent turns left 30 pixels and moves forward 20 pixels. Right actions means agent turns right 30 pixels and moves forward 20 pixels.

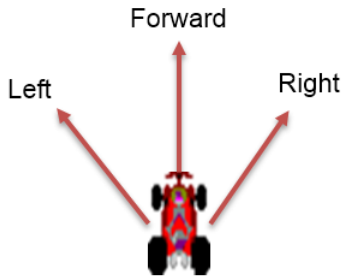


Figure 4.2: Three different actions in simulation

### 4.3.3 Measure of Performance: Estimated Energy Consumption on Software

To estimate energy consumption, duration and power consumption are needed. Duration is measured via python code. Power consumption is measured via LCD USB Mini Voltage and Current and capacity Detector USB Charger Tester Meter. This device can measure voltage and current at the same time while the robot is moving. Multiplying voltage and current is power. With time for each action and power from the device, the robot can estimate energy consumption for each action.

Duration for Forward is 1.5 seconds and duration for Left and Right is 1.9 seconds. Du-

ration for not actions is 0.7 seconds. Power consumption of performing action is 2.95W, and power consumption of not performing action is 2.30W. Therefore, estimated total energy consumption is:

$$\begin{aligned}\text{Estimated Total Energy Consumption} \\ \text{for performing action} \\ &= 2.95 \times 1.5 \times TotalNumberofForward \\ &+ 1.9 \times (TotalNumberofTurns) \end{aligned} \tag{4.1}$$

$$\begin{aligned}\text{Estimated Total Energy Consumption} \\ \text{for not performing action} \\ &= 0.7 \times 2.30 \times TotalNumberofActions \end{aligned} \tag{4.2}$$

$$\begin{aligned}\text{Estimated Total Energy Consumption} \\ &= (4.1) + (4.2) \end{aligned} \tag{4.3}$$

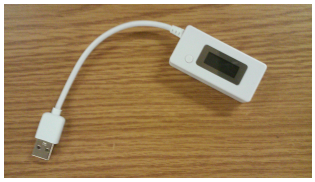

Figure 4.3: LCD usb mini voltage and current and capacity detector USB charger tester meter

### 4.4 Hardware Set-up, States, and Actions

The wall height is 30cm. Single sided corrugated cardboard is attached on the walls. With Single sided corrugated cardboard, ultrasonic sensor reflection problem can be solved(See Figure 3.1.1). Polystyrene foam board insulation is used for set-up walls. Electrical tape is used for the target which is the red target on software. With these, four different maps were created for testing the testbed. The relationship between software distances and hardware distance is that software distances multiply 30 and divides by 100. For example, 30 cm is same as 100 pixels.

#### 4.4.1 Sensor-Space Division

Distance thresholds are defined to discrete states in the given Table 4.2. Unlike software, if measured distance value is too large and not reasonable, state is same as short-range. During demonstration of the robot, if the value was extremely large or not reasonable, the robot was too close to the walls. This happens because of ultrasonic sensors'limitations.

Table 4.2: Discreted distance on hardware

| Distance | Value Given by the Sensor (cm) |
|---|---|
| Too Long Range | Longer than 90 |
| Long-Range | 60 to 90 |
| Mid-Range | 30 to 60 |
| Short-Range | 0 to 30 |

If an ultrasonic sensor measures higher than 90 cm, state is same as short range. That's because when the robot is too close to obstacle and the obstacle is not precise sensor alignment.

### 4.4.2 Action-Set Construction: Forward Left Right

To measure a particular distance,(4.4) equation is required. The Boe-Bot speed is 23cm/s.[16]

$$\text{Servo run time} = \frac{\text{Boe Bot distance}}{\text{Boe Bot speed}} \tag{4.4}$$

For deciding servo run time, delay function is used in the arduino. If the delay is equal to 2220, the robot will travel for 2.22 seconds. In this case, the robot moves forward 51.06cm. However, if the method is used for the thesis, the robot would not move correctly. Variables like battery power level, friction on different surfaces, and even manufacturing tolerances in the motors can severely reduce the robots accuracy in driving a particular distance.

Thats why digital encoder is used for the thesis.[16] The digital encoder provides wheel rotation feedback that the robot can use to perform more consistent maneuvers. The wheel diameter is 66.20mm, so circumference is 208mm. The wheel has 32 pokes, separated by 32 spaces, so the total of tick is 64. Therefore, one tick is same as move 3.25mm. For example, if the agent wants to move forward 32.5mm, the digital encoder should detect 10 ticks. Using the method, the agent can move exactly a particular distance.[16]

For rotating a particular angle, parallax tutorial table is used. If the robot detects 3 ticks, the robot rotates around 10 degrees. If the robot detects 17 ticks, the robot rotates 60 degrees. In the study, the robot should rotate 30 degree like software simulation. 8 ticks is the same as rotating 30 degrees. The finite set of the actions is defined by three discrete robot actions like the software's agent. Three basic actions: move forward (F), turn left and move forward, and turn right and move forward. The robot has the three actions based on

the robot heading orientation. Forward action means the robot moves forward 5cm. Left actions means agent turns left 30 degree and moves forward 5cm. Right actions means agent turns right 30 degree and moves forward 5cm.

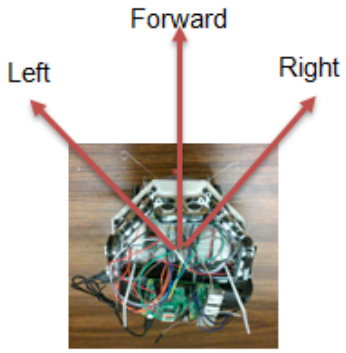


Figure 4.4: Hardware three actions

### 4.4.3 Measure of Performance Estimated Energy Sonsumption on Hardware

Duration for Forward is 1.5 seconds and duration for Left and Right is 1.9 seconds. Duration for no actions is 0.7 seconds. Power consumption of performing action is 2.95W, and power consumption of not performing action is 2.30W.

Estimated Total Energy Consumption

for performing action

$$= 2.95 \times 1.5 \times TotalNumberofForward$$
$$+ 1.9 \times (TotalNumberofTurns) \quad (4.5)$$

Estimated Total Energy Consumption

for not performing action

$$= 0.7 \times 2.30 \times Total Number of Actions \quad (4.6)$$

However, there is a difference between total measured duration and estimated total duration and the total measured duration is always longer than the total estimated duration. That's because raspberry pi has low CPU. Raspberry pi CPU is only 800MHz. For this reason, it generates the difference time.

$$\text{Estimated Extra Energy Consumtion} = \text{Difference duration} \times 2.30 \quad (4.7)$$

Therefore, for estimating total energy consumption on real robot, estimated total energy consumption is

$$\text{Estimated Total Energy Consumption} = (4.5) + (4.6) + (4.7) \quad (4.8)$$

## 4.5 Energy-Efficient Q-Learning

There are important three factors in Q-Learning: learning rate($\alpha$), discount factor($\gamma$) and reward function(R(s)). Q-Learning update function has the three factors. For t = 0,1,2,...

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha[R(s) + \gamma \max_a Q(s_{t+1}, a)] \quad (4.9)$$

An agent decides learning rate for how much new value will override the old information, and discount factor for the importance of future rewards. Reward function is given

depending on an action. For deciding learning rate $(\alpha)$ and discount factor $(\gamma)$, software simulation was tested on map 1. Based on the results, when learning rate is 0.1 and discount factor is 0.9, the total energy consumption is the smallest on map 1. Therefore, these values are used for the other maps.

Each action takes different energy consumption, so all actions cannot be same priority. Reward function can change priority for each action. Therefore, to give different priority to each action, reward function is modified.

### 4.5.1 Designed Reward Function

For giving different priority for each action, a reward function is modified. Forward action needs about 1.5 seconds, left action right action need about 1.9 seconds. Moreover power consumption for performing action is 2.95W. Based on these values, estimated energy consumption for forward action is 4.425J and estimated energy consumption for left and right action is 5.605J. Therefore, the ratio of forward, left and right actions is 1 : 1.26 : 1.26. Based on this relationship, the thesis used same ratio as energy consumption of each action reward. More energy consumption, more negative reward, so each action reward is negative.

- Duration for each action
  - Forward : 1.5 seconds
  - Left, Right : 1.9 seconds
- Power Consumption for performing action
  - 5.09(Voltage) * 0.58(Current) = 2.95W
- Estimated Energy Consumption
  - Forward Action Estimated Energy Consumption : 4.425J

  - Left Action Estimated Energy Consumption : 5.605J
  - Right Action Estimated Energy Consumption : 5.605J

- Reward Function Ratio
  - Reward(Forward) : Reward(Left),Reward(Right) = -1 : $-(1.26)^n$

When n is increased, total energy consumption is decreased. When n is 10, total energy consumption is the smallest on map 1.

### 4.5.2 Discount Factor

Discount factor means that an agent determines the important of future rewards. Therefore, the Discount factor can affect to total energy consumption. However, other researchers did not consider discount factor for their researches, so they did not try an experiment with varying discount factor and they used 0.9 value for discount factor. For example, Handy Wicaksono used 0.9 value for discount factor, and the author did not consider it for Q-learning behavior on autonomous navigation of physical robot [14]. Below Figure shows a results for the experiment with varying discount factor. Because of the below graph, discount factor value is 0.9 for the study.

### 4.5.3 Learning Rate

Learning rate means that how much new information will supersede an old information. Learning rate value is 0.1 for the thesis. If learning rate value is increased, the successful rate for convergence for desired performance is decreased. Therefore, 0.1 value was used for learning rate in this thesis.

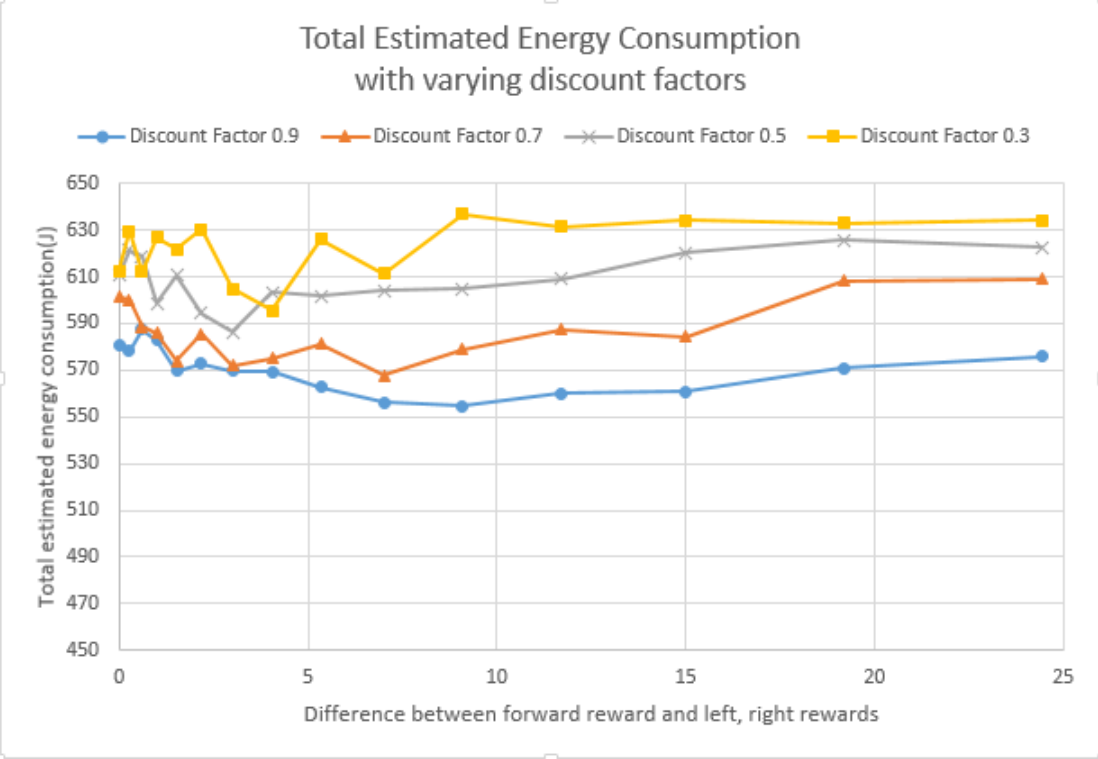


Figure 4.5: Estimated energy consumption with varying discount factor

## 5.1 Software

Ten different maps are used for the software experiments. The red square is a target, and black line is obstacles. The small car image is the agent. For collecting data, window 8 was used instead of raspberry pi. That's because window 8 is much faster than raspberry pi. The total iteration is 3000, discount factor is 0.9, and learning rate is 0.1. Twenty trials are needed for each maps, and calculate estimated energy consumption for each map.

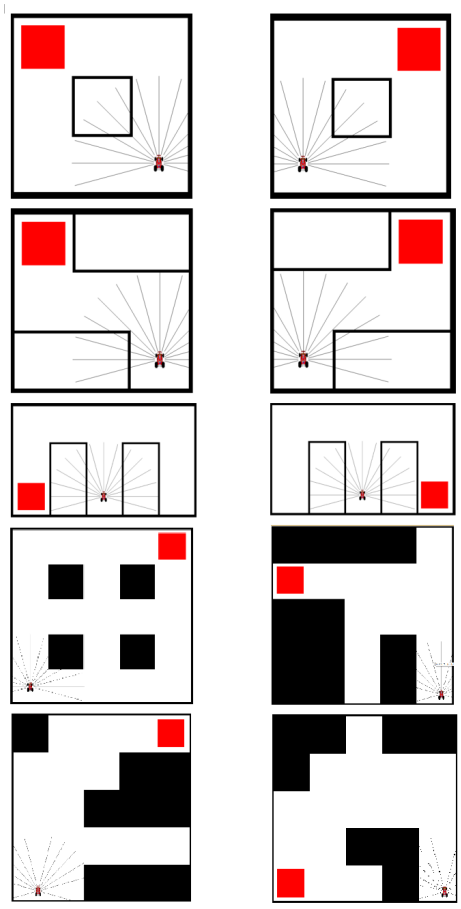

Figure 5.1: Ten different maps for simulation

## 5.2 Software Result

For testing Q-learning and energy-efficient Q-learning, the ten different maps are used for investigating various environments. For each map, Q-learning and energy-efficient Q-learning trajectories are compared. Also, total energy consumption, total duration and total movement distance are compared for these two Q-learning.

### 5.2.1 Software Map 1

This is a simple map. There is one obstacle in the map, and there is one red target on the top left. The agent only needs to turn left once to arrive at the red target. Below Figure 5.2 shows trajectory for the agent movement until it arrives at the red target.

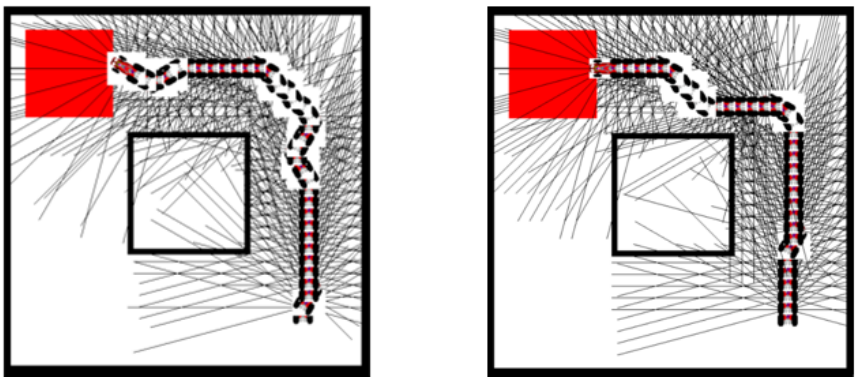

Figure 5.2: Software map 1 trajectory for Q-learning (left) and energy-efficient Q-learning (right)

### 5.2.2 Software Map 2

This is also a simple map like map 1. The agent only needs to turn right once to arrive at the red target. Below Figure 5.3 shows trajectory for the agent movement until it arrives at the red target. Like map 1, energy-efficient Q-learning algorithm is shorter than Q-learning.

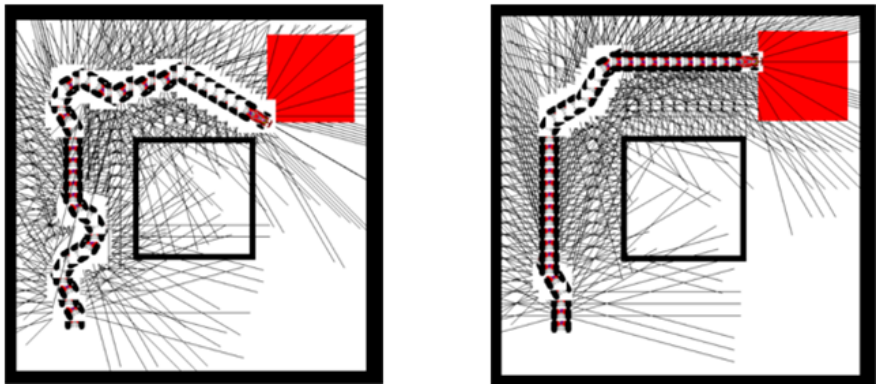

Figure 5.3: Software map 2 trajectory for Q-learning (left) and energy-efficient Q-learning (right)

### 5.2.3 Software Map 3

This is more complicated map than previous two maps. There are two obstacles in the map, and the map has one red target. The agent needs to turn left first and turn right to arrive at the red target. In this map, the agent used less left and right turns, so the trajectory is shorter than Q-learning.

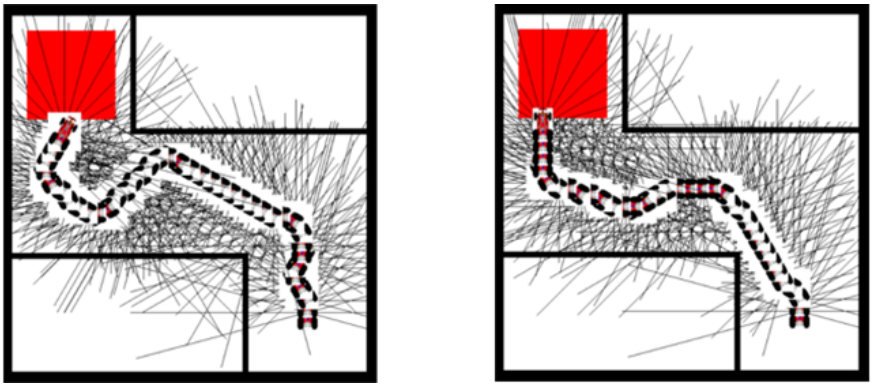

Figure 5.4: Software map 3 trajectory for Q-learning (left) and energy-efficient Q-learning (right)

### 5.2.4 Software Map 4

This map is similar to map 3. Unlike map 3, the agent needs to turn right first and turn left to arrive at the red target. However, the result is similar as map 3. The energy-efficient Q-learning trajectory is shorter, and it consumes less total duration. Therefore, the energy-efficient Q-learning uses less total energy consumption than the Q-learning.

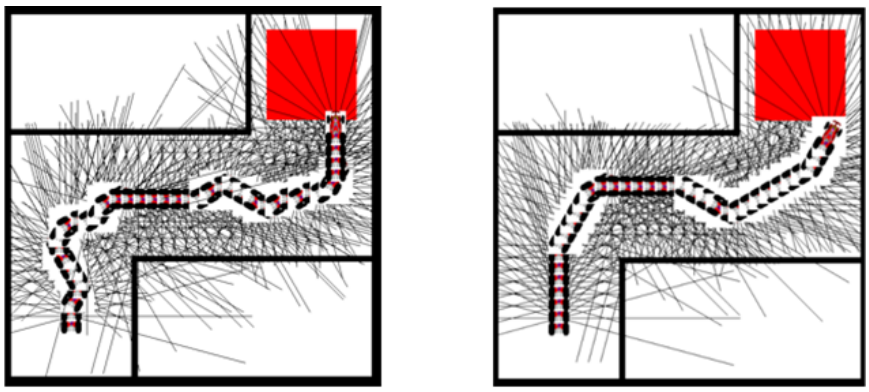

Figure 5.5: Software map 4 trajectory for Q-learning (left) and energy-efficient Q-learning (right)

### 5.2.5 Software Map 5

Through map 1 to map 4, the agent does not need to choose a path. However, the agent needs to choose a path to arrive at the red target. In this map, the red target is on the left side. During learning phase, the agent finds out where the target is. After the agent learns, the agent knows where the target is, and it moves to the target like below figure. This result shows the agent can detect the red target via Q-learning and energy-efficient Q-learning.

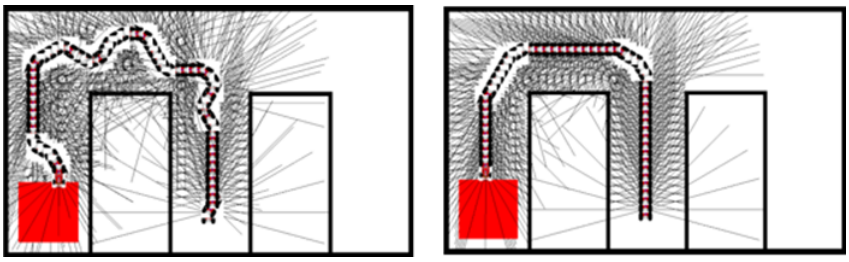

Figure 5.6: Software map 5 trajectory for Q-learning (left) and energy-efficient Q-learning (right)

### 5.2.6 Software Map 6

This map is same as map 5, but the target is on the right side. Therefore, the agent needs to choose path to arrive at the red target like map 5. In this map, even though the target is the other side, total energy consumption, total duration, and total agent's movement are smaller in energy-efficient Q-learning.

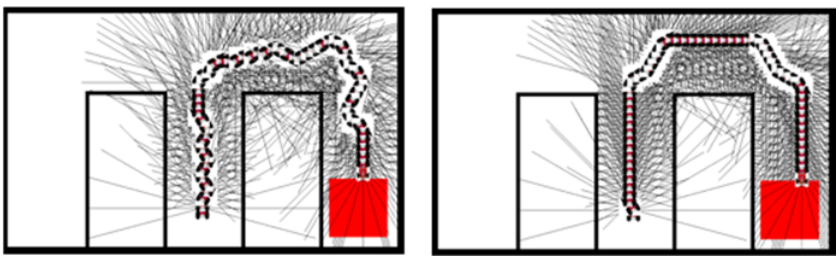

Figure 5.7: Software map 6 trajectory for Q-learning (left) and energy-efficient Q-learning (right)

### 5.2.7 Software Map 7

The agent needs to avoid four obstacles to arrive at the red target. There are several ways to arrive at the destination. In the map, the agent turns right only once in both Q-learning in simulation, but energy-efficient Q-learning algorithm takes less total energy

consumption, total duration, and total distance to arrive at the destination.

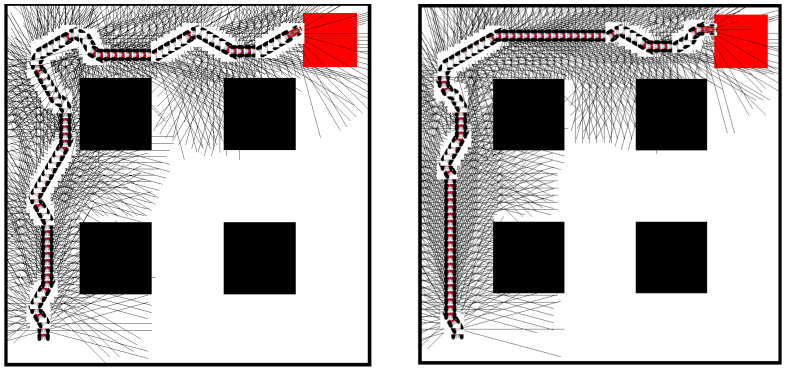

Figure 5.8: Software map 7 trajectory for Q-learning (left) and energy-efficient Q-learning (right)

### 5.2.8 Software Map 8

In this map, there are three big obstacles, and there is on one red target. The agent avoids the obstacles and arrives at the red target. The agent learns the target location during learning phase. After learning, the agent can arrive at the destination. Below figure is one of samples from software simulations for each Q-learning algorithms.

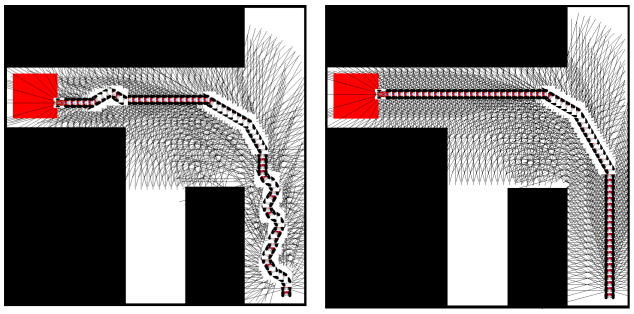

Figure 5.9: Software map 8 trajectory for Q-learning (left) and energy-efficient Q-learning (right)

### 5.2.9 Software Map 9

The red target is on the top right. The agent avoids three obstacles and arrives at the destination. Below figure shows trajectories for Q-learning result and energy-efficient Q-learning result.

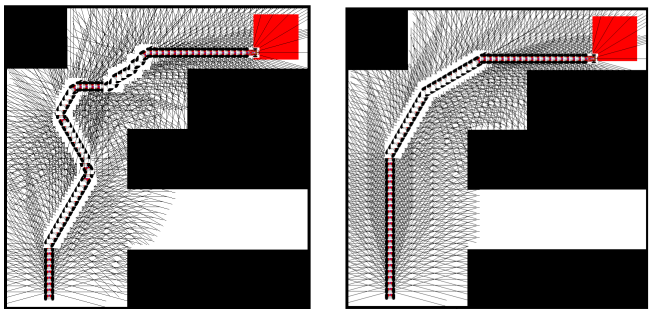

Figure 5.10: Software map 9 trajectory for Q-learning (left) and energy-efficient Q-learning (right)

### 5.2.10 Software Map 10

In this map, there are three obstacles and one red target. The agent needs to turn left at least two times to arrive at the red target.

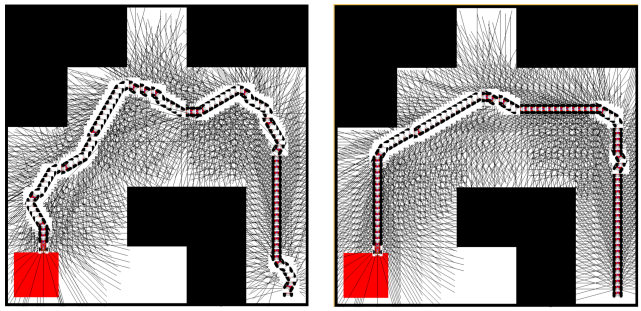

Figure 5.11: Software map 10 trajectory for Q-learning (left) and energy-efficient Q-learning (right)

### 5.2.11 Software Estimated Total Duration Result

Durations for each actions are measured manually. Forward duration is 1.5 seconds. Left and right durations are 1.9 seconds. Also the program counted the number of each actions. Based on these measurement and total number of actions, total duration can be estimated. The experimental results of comparison of the two different Q-Learning algorithm can be seen below Table.

Table 5.1: Software duration results on the various environments

| | Estimated Total Duration(s) | | Duration Saving(%) |
|---|---|---|---|
| Map | Q-Learning | Energy-Efficient Q-Learning | (Dnew Dold)/Dold*100 |
| Map1 | 60.8 | 57.7 | -5.09 |
| Map2 | 69.9 | 58.1 | -16.88 |
| Map3 | 67.7 | 59.8 | -11.66 |
| Map4 | 65.8 | 59.9 | -8.96 |
| Map5 | 103.8 | 86.2 | -16.95 |
| Map6 | 101.6 | 87.8 | -13.58 |
| Map7 | 190.8 | 178.3 | -6.55 |
| Map8 | 161.7 | 147.7 | -8.65 |
| Map9 | 179.8 | 147.2 | -18.1 |
| Map10 | 210.9 | 187.1 | -11.2 |

### 5.2.12 Software Estimated Total Distance Result

All actions'distances are same. The distance is 20 pixels on software. Based on the distance and the number of actions, total distance can be estimated. The experimental results of comparison of the two different Q-Learning algorithm can be seen below Table.

Table 5.2: Software distance results on the various environments

| | Estimated Total Distance(pixels) | | Distance Saving(%) |
|---|---|---|---|
| Map | Q-Learning | Energy-Efficient Q-Learning | (Dnew Dold)/Dold*100 |
| Map1 | 700 | 680 | -2.85 |
| Map2 | 780 | 700 | -10.25 |
| Map3 | 760 | 680 | -10.52 |
| Map4 | 740 | 700 | -8.96 |
| Map5 | 1200 | 1040 | -16.95 |
| Map6 | 1120 | 1060 | -13.58 |
| Map7 | 1635 | 1561 | -4.48 |
| Map8 | 1401 | 1320 | -5.78 |
| Map9 | 1578 | 1318 | -16.4 |
| Map10 | 1795 | 1666 | -7.18 |

### 5.2.13 Software Estimated Total Energy Consumption Result

Based on the measurements, power consumptions and the number of actions, total energy consumptions are estimated for the various maps. The experimental results of comparison of the two different Q-Learning algorithm can be seen below Table.

Table 5.3: Software energy consumption results on the various environments

| | Estimated Energy Consumption(J) | | Energy Saving(%) |
|---|---|---|---|
| Map | Q-Learning | Energy-Efficient Q-Learning | (Enew Eold)/Eold*100 |
| Map1 | 230.105 | 218.170 | -5.18 |
| Map2 | 264.865 | 219.485 | -17.13 |
| Map3 | 256.470 | 226.430 | -11.71 |
| Map4 | 249.255 | 226.565 | -9.10 |
| Map5 | 392.780 | 325.620 | -17.09 |
| Map6 | 385.160 | 331.655 | -13.89 |
| Map7 | 525.737 | 490.702 | -6.66 |
| Map8 | 455.274 | 405.819 | -10.8 |
| Map9 | 494.687 | 404.314 | -18.2 |
| Map10 | 581.466 | 514.161 | -11.5 |

## 5.3 Hardware

The testbed tried on four different maps. Hardware map 1 through map 4 are same as software map 1 through map 4. This results show energy-efficient Q-learning works on the real world.

Figure 5.12: Four different maps on the real world

## 5.4 Hardware Result

For testing q-learning and energy efficiency q-learning on the real world, four different maps are used for investigating various environments. First two maps are simple map, the agent needs to turn once. Map 3 and map 4, the agent needs to turn left and right once respectively.

### 5.4.1 Hardware Map 1

This is a simple map. The robot only needs to turn left once to arrive at the electrical tape, and it is same as software map 1.

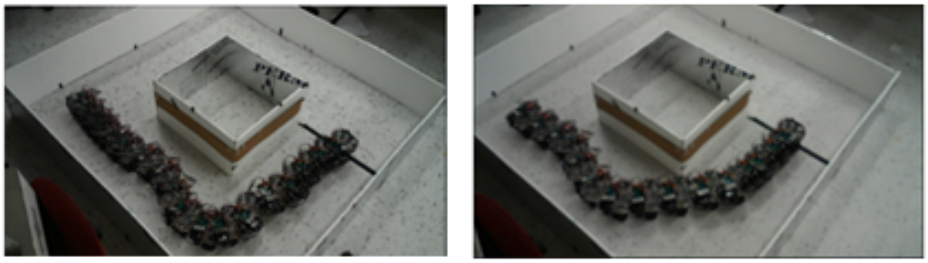

Figure 5.13: Hardware map 1 trajectory for Q-learning(left) and energy-efficient Q-learning(right)

### 5.4.2 Hardware Map 2

This is a simple map. The robot only needs to turn right once to arrive at the electrical tape, and it is same as software map 2.

Figure 5.14: Hardware map 2 trajectory for Q-learning(left) and energy-efficient Q-learning(right)

### 5.4.3 Hardware Map 3

This is more complicated map than previous two maps. The robot needs to turn left first and turn right to arrive at the electrical tape, and it is same as software map 3.

Figure 5.15: Hardware map 3 trajectory for Q-learning(left) and energy-efficient Q-learning(right)

### 5.4.4 Hardware Map 4

This is more complicated map than previous two maps. The robot needs to turn right first and turn left to arrive at the electrical tape, and it is same as software map 4.

Figure 5.16: Hardware map 4 trajectory for Q-learning(left) and energy-efficient Q-learning(right)

### 5.4.5 Hardware Total Duration Result

Total duration is actually measured, not estimated. Total duration is until the robot arrives at the destination, which is a black tape line. The results of comparison of the two algorithm can be seen below Table.

Table 5.4: Hardware total duration results on the various environments

| | Total Duration(s) | | Duration Saving(%) |
|---|---|---|---|
| Map | Q-Learning | Energy-Efficient Q-Learning | (Dnew Dold)/Dold*100 |
| Map1 | 96 | 85 | -11.45 |
| Map2 | 96 | 92 | -4.16 |
| Map3 | 113 | 111 | -1.77 |
| Map4 | 120 | 965 | -20.0 |

### 5.4.6 Hardware Estimated Total Distance Result

All actions'distances are same. Each action's distance is about 5 cm. Based on it and the number of each actions, total distance is estimated. The results of comparison can be seen below Table.

Table 5.5: Hardware total distance results on the various environments

| | Total Distance(cm) | | Distance Saving(%) |
|---|---|---|---|
| Map | Q-Learning | Energy-Efficient Q-Learning | (Dnew Dold)/Dold*100 |
| Map1 | 155 | 140 | -9.67 |
| Map2 | 160 | 150 | -6.25 |
| Map3 | 190 | 180 | -5.26 |
| Map4 | 200 | 155 | -22.5 |

### 5.4.7 Hardware Estimated Total Energy Consumption Result

Based on measurements, power consumption and the number of each actions, total energy consumptions are estimated. The energy-efficient Q-Learning consumes less energy than Q-Learning on the real robot. The experimental results of comparison of the two different Q-Learning algorithm on the robot can be seen Table.

Table 5.6: Hardware result on the various environments

| | Estimated Energy Consumption(J) | | Energy Saving(%) |
|---|---|---|---|
| Map | Q-Learning | Energy-Efficient Q-Learning | (Enew Eold)/Eold*100 |
| Map1 | 257.265 | 224.620 | -12.68 |
| Map2 | 256.940 | 245.010 | -4.64 |
| Map3 | 301.890 | 295.340 | -2.16 |
| Map4 | 322.540 | 254.405 | -21.10 |

# 6. CONCLUSION

This master's thesis has shown that a real robot can avoid collision with energy-efficient Q-learning. In this thesis, some background on Q-learning was described in the beginning. The testbed description was explained, and all parts were specified in detail. Next, upgraded energy-efficient Q-learning is interpreted. Reward function was adjusted for energy efficiency Q-Learning. For improving energy consumption on Q-Learning, energy-efficient Q-Learning was designed. To achieve this goal, each action had various rewards to get different priorities. The other factors were fixed, and these values were determined via simulations. After the energy-efficient Q-learning explanation, using python standard library, the energy-efficient Q-Learning was simulated. The ten different maps were tested for original Q-learning and energy efficiency Q-learning on simulation. Furthermore, the energy-efficient Q-Learning was also tested on a real robot. Four different maps were used for testing original Q-learning and energy-efficient Q-learning on the testbed.

On the software, the code in python is designed as hardware-friendly, so the results are similar as hardware results. Moreover, the real robot can use Q-Table that produce via simulation because the code is hardware-friendly. The hardware is tested and verified energy-efficient Q-Learning for collision avoidance.

The energy-efficient Q-learning algorithm consumes less energy than original Q-learning on simulation. The agent requires rotating left or right once on map 1 and 2. The agent requires rotating one left turn and one right turn on map 3 and 4. The agent has the options to choose direction left or right on map 5 and 6. A 5.18%, 17.13%, 11.71%, 9.10%, 17.09%, and 13.89% reduction is performed for map 1 through 6 ,respectively. The same map 1 through 4 were investigated with the testbed. On the real robot, a 12.68%, 4.64%, 2.16%

and 21.10% reduction is also reached, on map 1 through 4, respectively on the testbed. Significant energy efficiency improvement has been achieved for the simulation and the demonstrating testbed.

To summarize, a real robot with simulation Q-Table was seen to perform the task of obstacle avoidance by using energy efficiency Q-learning. The energy-efficient Q-learning shows not only smaller energy-efficient than Q-learning but also better trajectory path for the agent. In the future, energy-efficient Q-Learning will be investigated doing more various actions and more complicated maps.

# REFERENCES


[1] R. S. Sutton, *Reinforcement Learning: Introduction*. Springer US, 1998.

[2] T. J. Appel, "The development of a robotic test bed with applications in q-learning," Master's thesis, Kettering University, 2009.

[3] O. Liang, "Raspberry pi and arduino connected using i2c - oscar liang." [Online]. Available : https://oscarliang.com/raspberry-pi-arduino-connected-i2c/, 2013-2017.

[4] "Automatically connect a raspberry pi to a wifi network." [Online]. Available : http://weworkweplay.com/play/automatically-connect-a-raspberry-pi-to-a-wifi-network/.

[5] J. Han, "Robot-aided learning and r-learning services," *Department of Computer Education Cheongju National University of Education*, 2010.

[6] C. XIA, *Intelligent Mobile Robot Learning in Autonomous Navigation*. PhD thesis, Beihang University, 2015.

[7] K. B. Lazhar Khriji, Farid Touati and A. Al-Yahmedi, "Mobile robot navigation based on q-learning technique," *International Journal of Advanced Robotic Systems*, 2011.

[8] Z. Z. Lvwen Huang, Dongjian He and P. Zhang, "Autonomous navigation based on a q-learning algorithm for a robot in a real environment," *Intelligent Automation & Soft Computing*, pp. 317–323, 2016.

[9] T. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[10] Y. M. Eyal Even-Dar, "Learning rates for q-learning," *Journal of Machine Learning Research 5*, 2003.

[11] M. Tokic and G. Palm, "Value-difference based exploration: Adaptive control between epsilon-greedy and softmax," *Advances in Artificial Intelligence of Lecture Notes in Artificial Intelligence,*, 2011.

[12] J. Blynel, "Reinforcement learning on real robots," Master's thesis, University of Aarhus, 2000.

[13] F. S. Clement Strauss, "Autonomous navigation based on a q-learning algorithm for a robot in a real environment," *Rochester Institute of Technology RIT Scholar Works*, 2008.

[14] H. Wicaksono, "Q learning behavior on autonomous navigation of physical robot," *The 8th International Conference on Ubiquitous Robots and Ambient Intelligence*, 2011.

[15] I. Idris, *Instant Pygame for Python Game Development How-to*. Packt Publishing, 2013.

[16] "Activity 3: Calculating distances learn.parallax.com." [Online]. Available : http://learn.parallax.com/tutorials/robot/shield-bot/robotics-board-education-shield-arduino/chapter-4-boe-shield-bot-11, 2017.

APPENDIX A

## A.1 Software Learning Phase Python Code

```
#Seungjai Ahn
#Collecet Data
#Map 1,2,3,4,5,6
#Left Right Reward
#Iteration 3000

import warnings
warnings.filterwarnings('ignore')
import learning, pygame, time, sys, Sprite_New_SJ
import math
import random
import cPickle
sys.setrecursionlimit(15000)

pygame.init()

# DISPLAY CONSTANTS
fps             = 60
display_width   = 810
display_height  = 810
```

```
car_width       = 35
car_height      = 35
black           = (0,0,0)
white           = (255,255,255)
red             = (255,0,0)
blue            = (0,0,255)
green           = (0,255,0)

# DISPLAY SETTING IN PYGAME
gameDisplay = pygame.display.set_mode(
(display_width, display_height))
pygame.display.set_caption('Car␣Simulator_Q')
clock = pygame.time.Clock()

# Q-LEARNING
class static:
    FLAG = 1
    REWARD = 0

class Obstacle:      #for now just colored rectangles
    def __init__(self, x, y, width, height, color):
        self.x          = x
        self.y          = y
        self.width      = width
        self.height     = height
```

```
        self.color      = color

#Map 1
list_obstacles = []
w01 = Obstacle(0,0,810,105,black)    #top wall
list_obstacles.append(w01)
w02 = Obstacle(705,0,105,810,black) #right wall
list_obstacles.append(w02)
w03 = Obstacle(0,705,810,105,black) #bottom wall
list_obstacles.append(w03)
w04 = Obstacle(0,0,105,810,black)    #left wall
list_obstacles.append(w04)

w05 = Obstacle(305,305,10,200,black)
list_obstacles.append(w05)
w07 = Obstacle(305,305,200,10,black)
list_obstacles.append(w07)
w06 = Obstacle(505,305,10,200,black)
list_obstacles.append(w06)
w08 = Obstacle(305,505,210,10,black)
list_obstacles.append(w08)
list_target = []
w09=Obstacle(130,130,150,150,red) #target
list_target.append(w09)
```

```
#Map 2
#list_obstacles = []
#w01 = Obstacle(0,0,810,105,black)    #top wall
#list_obstacles.append(w01)
#w02 = Obstacle(705,0,105,810,black) #right wall
#list_obstacles.append(w02)
#w03 = Obstacle(0,705,810,105,black) #bottom wall
#list_obstacles.append(w03)
#w04 = Obstacle(0,0,105,810,black)    #left wall
#list_obstacles.append(w04)
#
#w05 = Obstacle(305,305,10,200,black)
#list_obstacles.append(w05)
#w07 = Obstacle(305,305,200,10,black)
#list_obstacles.append(w07)
#w06 = Obstacle(505,305,10,200,black)
#list_obstacles.append(w06)
#w08 = Obstacle(305,505,210,10,black)
#list_obstacles.append(w08)
#list_target = []
#w09=Obstacle(535,130,150,150,red) #target
#list_target.append(w09)

#Map 3
#list_obstacles = []
```

```
#w01 = Obstacle(0,0,810,105,black)    #top wall
#list_obstacles.append(w01)
#w02 = Obstacle(705,0,105,810,black) #right wall
#list_obstacles.append(w02)
#w03 = Obstacle(0,705,810,105,black) #bottom wall
#list_obstacles.append(w03)
#w04 = Obstacle(0,0,105,810,black)    #left wall
#list_obstacles.append(w04)
#
#w05 = Obstacle(305,105,10,200,black)
#list_obstacles.append(w05)
#w06 = Obstacle(305,295,400,10,black)
#list_obstacles.append(w06)
#w07 = Obstacle(105,505,400,10,black)
#list_obstacles.append(w07)
#w08 = Obstacle(495,505,10,200,black)
#list_obstacles.append(w08)
#
#list_target = []
#w09=Obstacle(130,130,150,150,red) #target
#list_target.append(w09)

#Map 4
#list_obstacles = []
#w01 = Obstacle(0,0,810,105,black)    #top wall
```

```
#list_obstacles.append(w01)
#w02 = Obstacle(705,0,105,810,black) #right wall
#list_obstacles.append(w02)
#w03 = Obstacle(0,705,810,105,black) #bottom wall
#list_obstacles.append(w03)
#w04 = Obstacle(0,0,105,810,black)    #left wall
#list_obstacles.append(w04)
#
#w05 = Obstacle(495,105,10,200,black)
#list_obstacles.append(w05)
#w06 = Obstacle(105,295,400,10,black)
#list_obstacles.append(w06)
#w07 = Obstacle(305,505,400,10,black)
#list_obstacles.append(w07)
#w08 = Obstacle(305,505,10,200,black)
#list_obstacles.append(w08)
#list_target = []
#w09=Obstacle(530,130,150,150,red) #target
#list_target.append(w09)

#Map 5
#list_obstacles = []
#w01 = Obstacle(0,0,1210,105,black)    #top wall
#list_obstacles.append(w01)
#w02 = Obstacle(1105,0,105,810,black) #right wall
```

```
#list_obstacles.append(w02)
#w03 = Obstacle(0,705,1210,105,black) #bottom wall
#list_obstacles.append(w03)
#w04 = Obstacle(0,0,105,810,black)    #left wall
#list_obstacles.append(w04)
#
#w05 = Obstacle(305,305,10,400,black)
#list_obstacles.append(w05)
#w07 = Obstacle(305,305,200,10,black)
#list_obstacles.append(w07)
#w06 = Obstacle(505,305,10,400,black)
#list_obstacles.append(w06)
#
#w08 = Obstacle(705,305,10,400,black)
#list_obstacles.append(w08)
#w10 = Obstacle(705,305,200,10,black)
#list_obstacles.append(w10)
#w11 = Obstacle(905,305,10,400,black)
#list_obstacles.append(w11)
#
#
#list_target = []
#w09=Obstacle(130,530,150,150,red) #target
#list_target.append(w09)
```

```
#Map 6
#list_obstacles = []
#w01 = Obstacle(0,0,1210,105,black)    #top wall
#list_obstacles.append(w01)
#w02 = Obstacle(1105,0,105,810,black) #right wall
#list_obstacles.append(w02)
#w03 = Obstacle(0,705,1210,105,black) #bottom wall
#list_obstacles.append(w03)
#w04 = Obstacle(0,0,105,810,black)    #left wall
#list_obstacles.append(w04)
#
#w05 = Obstacle(305,305,10,400,black)
#list_obstacles.append(w05)
#w07 = Obstacle(305,305,200,10,black)
#list_obstacles.append(w07)
#w06 = Obstacle(505,305,10,400,black)
#list_obstacles.append(w06)
#
#w08 = Obstacle(705,305,10,400,black)
#list_obstacles.append(w08)
#w10 = Obstacle(705,305,200,10,black)
#list_obstacles.append(w10)
#w11 = Obstacle(905,305,10,400,black)
#list_obstacles.append(w11)
#
```

```
#
#list_target = []
#w09=Obstacle(930,530,150,150,red) #target
#list_target.append(w09)

#QLearning SETTING
class agent:
    count = 0
    n_arr = 0
    n_f = 0
    n_r1 = 0
    n_l1 = 0
    n_r2 = 0
    n_l2 = 0
    p_c = 0
    n_moving = 0
    rewards = 0 #Reward
    epsilon = 1.0 #Epsilon
    alpha = 0.1 #Learning Rate
    gamma = 0.9 #Discount Factor
    iteration = 3000
    count_test = 0
    iter_test = 20
    n_t_arr = 0
#initialized Q-Table
```

```
Q = []
total_reward = 0
#Forward = 1
#Left = 2
#Right = 3
ACTIONLIST = [1,2,3]
lastState = ()
lastAction = static.FLAG

carAgent=learning.QLearning(ACTIONLIST,
agent.epsilon, agent.alpha, agent.gamma)

def move():
    #static.REWARD = -1
    update()
    agent.rewards += static.REWARD
    static.REWARD = 0

# Q LEARNING UPDATE
def update():
    temp = []
    state = temp
    s = car.sense(list_rect_obstacles)
    list_temp = s
```

```
#Ultrasonic sensor 1
list_temp2=[0,0,0]
    min_value1 = min(list_temp[0:2])
    if(0 <= min_value1 < 100):
        list_temp2[0] = 0
    elif(100 <= min_value1 < 200):
        list_temp2[0] = 1
    elif(200 <= min_value1 < 299):
        list_temp2[0] = 2
    else:
        list_temp2[0] = 3

#Ultrasonic sensor 2
    min_value2 = min(list_temp[6:8])
    if(0 <= min_value2 < 100):
        list_temp2[1] = 0
    elif(100 <= min_value2 < 200):
        list_temp2[1] = 1
    elif(200 <= min_value2 < 299):
        list_temp2[1] = 2
    else:
        list_temp2[1] = 3

#Ultrasonic sensor 3
    min_value3 = min(list_temp[12:14])
```

```
if(0 <= min_value3 < 100):
    list_temp2[2] = 0
elif(100 <= min_value3 < 200):
    list_temp2[2] = 1
elif(200 <= min_value3 < 299):
    list_temp2[2] = 2
else:
    list_temp2[2] = 3

if(s_degs == 4):
    s_degs = 0

temp = list_temp2
# since distance [] is of no use
if temp != []:
    state = temp
    state = tuple(state)
    #action = static.FLAG
    #Q.append([state, action])
    lastState = Q[len(Q)-1][0]
    lastAction = Q[len(Q)-1][1]

carAgent.learn(lastState, lastAction,
               static.REWARD, state)
static.FLAG = carAgent.chooseAction(state)
```

```
    action = static.FLAG
    Q.append([state, action])

# GAME SIMULATION LOOP
def game_loop():
    deltat = clock.tick(fps)
    gameDisplay.fill(white)
    pygame.init()
    if(agent.count < agent.iteration):

        agent.count += 1
        if agent.count == 300:
            carAgent.epsilon = 0.9
        elif agent.count == 600:
            carAgent.epsilon = 0.8
        elif agent.count == 900:
            carAgent.epsilon = 0.7
        elif agent.count == 1200:
            carAgent.epsilon = 0.6
        elif agent.count == 1500:
            carAgent.epsilon = 0.5
        elif agent.count == 1800:
            carAgent.epsilon = 0.4
        elif agent.count == 2100:
            carAgent.epsilon = 0.3
```

```
        elif agent.count == 2400:
            carAgent.epsilon = 0.2
        elif agent.count == 2700:
            carAgent.epsilon = 0.1


        agent.n_f = 0
        agent.n_r1 = 0
        agent.n_l1 = 0
        agent.n_moving = 0
        agent.p_c = 0
        print (agent.count)

    else:#TEST count
        carAgent.epsilon=0


        agent.n_f = 0
        agent.n_r1 = 0
        agent.n_l1 = 0
        agent.n_moving = 0
        agent.p_c = 0
        agent.count_test +=1
        if agent.count_test == 200:


            with \
            open('Updated_Collect_Data\
```

```
␣␣␣␣␣␣␣␣␣␣␣␣␣␣NewMap_1_DF3_RE06_01.txt',\
                    'w') \
                as savefile:
                    cPickle.dump(carAgent, savefile)
        if agent.count_test >=200:
            #Finish Simulation
            quit()
        print 'from␣Q-table'
        print agent.count_test


    global car, car_group
    #Start Point (car)
    #Map 1
    car = Sprite_New_SJ.Robot('car9.png', (605,605))
    #Map 2
    #car = Sprite_New_SJ.Robot('car9.png', (205,605))
    #Map 3
    #car=Sprite_New_SJ.Robot('car9.png',(605,605))
    #Map 4
    #car = Sprite_New_SJ.Robot('car9.png', (205,605))
    #Map 5,6
    #car = Sprite_New_SJ.Robot('car9.png', (605,605))
    car_group = pygame.sprite.RenderPlain(car)
    global list_rect_obstacles, list_rect_target
    list_rect_obstacles = []
```

```
for ob in list_obstacles:
    list_rect_obstacles.append\
    (pygame.draw.rect\
     (gameDisplay,black,(ob.x,\
                         ob.y,\
                         ob.width,\
                         ob.height)))
list_rect_target = []
for tar in list_target:
    list_rect_target.append\
    (pygame.draw.rect\
     (gameDisplay,red,(tar.x,\
                       tar.y,\
                       tar.width,\
                       tar.height)))
car_group.update(deltat)
car_group.draw(gameDisplay)
car.draw_rays(gameDisplay)
gameDisplay.blit(car.image, car.rect)
pygame.display.flip()

dx = car.x - w09.x
dy = car.y - w09.y
rads = (math.atan2(dx,dy))
degs = math.degrees(rads)
```

```
glo_degs = degs
dir_angle = (degs - car.direction)
if(dir_angle <0):
    dir_angle = 360 + dir_angle
s = car.sense(list_rect_obstacles)
list_temp = s
list_temp2=[0,0,0]

min_value1 = min(list_temp[0:2])
if(0 <= min_value1 < 100):
    list_temp2[0] = 0
elif(100 <= min_value1 < 200):
    list_temp2[0] = 1
elif(200 <= min_value1 < 299):
    list_temp2[0] = 2
else:
    list_temp2[0] = 3

min_value2 = min(list_temp[6:8])
if(0 <= min_value2 < 100):
    list_temp2[1] = 0
elif(100 <= min_value2 < 200):
    list_temp2[1] = 1
elif(200 <= min_value2 < 299):
    list_temp2[1] = 2
```

```
else:
    list_temp2[1] = 3


min_value3 = min(list_temp[12:14])
if(0 <= min_value3 < 100):
    list_temp2[2] = 0
elif(100 <= min_value3 < 200):
    list_temp2[2] = 1
elif(200 <= min_value3 < 299):
    list_temp2[2] = 2
else:
    list_temp2[2] = 3


s_degs = int((dir_angle+45)/90)
agent.p_c += 5*15*0.001*0.00001*3
if(s_degs == 4):
    s_degs = 0
agent.p_c += 7.2*42*0.001*0.00001
#list_temp.append(s_degs)
temp = list_temp2
# since distance [] is of no use


if temp != []:
    state = temp
    state = tuple(state)
```

```
        action = static.FLAG
        Q.append([state, action])

gameExit = False
# IN GAME
while not gameExit:
    # USER INPUT
    deltat = clock.tick(fps)
    gameDisplay.fill(white)

    # MEASURE DISTANCE with TARGET,
    # position of car, and speed of car
    global dist, dist1, car_position, \
    car_speed, dir_angle, glo_degs
    # RENDERING
    global list_rect_obstacles, list_rect_target
    list_rect_obstacles = []
    for ob in list_obstacles:
        list_rect_obstacles.append\
        (pygame.draw.rect(gameDisplay,black,\
                          (ob.x,\
                           ob.y,\
                           ob.width,\
                           ob.height)))
    list_rect_target = []
```

```
for tar in list_target:
    list_rect_target.append\
    (pygame.draw.rect(gameDisplay,red,\
                      (tar.x,\
                       tar.y,\
                       tar.width,\
                       tar.height)))
dx = car.x - w09.x
dy = car.y - w09.y
rads = (math.atan2(dx,dy))
degs = math.degrees(rads)
glo_degs = degs
dir_angle = (degs - car.direction)
if(dir_angle <0):
    dir_angle = 360 + dir_angle
for event in pygame.event.get():
    if not hasattr(event, 'key'): continue
    down = event.type == pygame.KEYDOWN
    if event.key == pygame.K_ESCAPE:
        txt.close()
        sys.exit(0)

time_f = 1.0
time_l1 = 1.3
time_r1 = 1.3
```

```
if(static.FLAG == 1):#Forward
    agent.n_moving += 1
    agent.n_f += 1
    static.REWARD = -1
    car.k_up = -20
    car.k_down  = 0
    car.k_left  = 0
    car.k_right = 0

elif(static.FLAG == 2):#Left
    agent.n_moving += 1
    agent.n_l1 += 1
    static.REWARD = -pow(1.26,0)
    car.k_up    = -20
    car.k_down  = 0
    car.k_left  = 30
    car.k_right = 0

elif(static.FLAG == 3):#Right
    agent.n_moving += 1
    agent.n_r1 += 1
    static.REWARD = -pow(1.26,0)
    car.k_up    = -20
    car.k_down  = 0
    car.k_left  = 0
```

```
        car.k_right = -30

    car_group.update(deltat)
    car_group.draw(gameDisplay)
    car.draw_rays(gameDisplay)
    gameDisplay.blit(car.image, car.rect)
    pygame.display.flip()

    #ARRIVAL
    if car.rect.collidelist\
    (list_rect_target) != -1:

        print 'arrive'
        if(agent.count<agent.iteration):
            agent.n_arr += 1
        if(agent.count_test >= 1 ):
            agent.n_t_arr= agent.n_t_arr + 1
        static.REWARD = 100
        lastState = Q[len(Q)-1][0]
        lastAction = Q[len(Q)-1][1]
        carAgent.q[(lastState, lastAction)] \
                    = static.REWARD
        static.REWARD = 0
        game_loop()
```

```
        #COLLISION
        elif (car.rect.collidelist\
              (list_rect_obstacles) != -1):
            print 'crash'
            print "#of success :%d" % agent.n_arr
            static.REWARD = -100
            lastState = Q[len(Q)-1][0]
            lastAction = Q[len(Q)-1][1]
            carAgent.q[(lastState, lastAction)] \
                        = static.REWARD
            agent.rewards += static.REWARD
            static.REWARD = 0
            game_loop()

        else:
            move()

static.FLAG = random.randint(1,3)
game_loop()
pygame.quit()
quit()
```

### A.2 Hardware Arduino Uno Code

```
#include <Servo.h>
#include <Wire.h>
#define SLAVE_ADDRESS 0x04
```

```
int number=0;
int state=0;
Servo servoLeft;
Servo servoRight;

int Rindex = 0;
int Lindex = 0;
int temp_val_l = 0;
int temp_val_r = 0;

int line_follower = 0;
byte wLeft = digitalRead(5);
byte wRight = digitalRead(7);
int flag_data = 0;
int flag_coll_des = 0;
void setup(){
  Serial.begin(9600);
  // initialize i2c as slave
  Wire.begin(SLAVE_ADDRESS);
  tone(4,3000,1000);
  delay(1000);
  //Digital Encoder
  pinMode(11,INPUT);
  pinMode(10,INPUT);
```

```
  //Bumper
  pinMode(7,INPUT);
  pinMode(5,INPUT);
  servoLeft.attach(13);
  servoRight.attach(12);
  // define callbacks for i2c communication
  servoLeft.writeMicroseconds(1500);
  servoRight.writeMicroseconds(1500);
  Wire.onReceive(receiveData);
  Wire.onRequest(sendData);
}

void loop(){
  if(number == 1){//Foward
    Rindex = 0;
    Lindex = 0;
    temp_val_l = digitalRead(11);
    temp_val_r = digitalRead(10);
    delay(500);
    forward();
    while(1){
      flag_coll_des = collision_destination();
      if(flag_coll_des != 1){
        break;
      }
```

```
      if(Rindex > 10 || Lindex > 10){//Forward 5 cm
        break;
      }
      if(temp_val_l != digitalRead(11)){
        Lindex++;
        temp_val_l = digitalRead(11);
      }
      if(temp_val_r != digitalRead(10)){
        Rindex++;
        temp_val_r = digitalRead(10);
      }
    }
    servoLeft.writeMicroseconds(1500);
    servoRight.writeMicroseconds(1500);
    number = 0;
  }else if(number == 2){
    Lindex = 0;
    Rindex = 0;
    temp_val_l = digitalRead(11);
    temp_val_r = digitalRead(10);
    delay(500);
    turnLeft();
    while(1){
      flag_coll_des = collision_destination();
      if(flag_coll_des != 1){
```

```
      break;
    }
    if(Rindex >= 3 && Lindex >= 3){
      break;
    }
    if(temp_val_l != digitalRead(11)){
      Lindex++;
      temp_val_l = digitalRead(11);
    }
    if(temp_val_r != digitalRead(10)){
      Rindex++;
      temp_val_r = digitalRead(10);
    }
  }
  servoLeft.writeMicroseconds(1500);
  servoRight.writeMicroseconds(1500);
  delay(500);
  Lindex = 0;
  Rindex = 0;
  temp_val_l = digitalRead(11);
  temp_val_r = digitalRead(10);
  forward();
  while(1){
    flag_coll_des = collision_destination();
    if(flag_coll_des != 1){
```

```
      break;
    }
    if(Rindex > 10||Lindex > 10){//Forward 5 cm
      break;
    }
    if(temp_val_l != digitalRead(11)){
      Lindex++;
      temp_val_l = digitalRead(11);
    }
    if(temp_val_r != digitalRead(10)){
      Rindex++;
      temp_val_r = digitalRead(10);
    }
  }
  servoLeft.writeMicroseconds(1500);
  servoRight.writeMicroseconds(1500);
  delay(500);
  number = 0;
}else if(number == 3){
  Serial.println("Right");
  Lindex = 0;
  Rindex = 0;
  temp_val_l = digitalRead(11);
  temp_val_r = digitalRead(10);
  delay(500);
```

```
turnRight();
while(1){
  flag_coll_des = collision_destination();
  if(flag_coll_des != 1){
    Serial.println("flag_coll_Dest");
    Serial.println(flag_coll_des);
    break;
  }
  if(Rindex >= 3 && Lindex >= 3){
    break;
  }
  if(temp_val_l != digitalRead(11)){
    Lindex++;
    temp_val_l = digitalRead(11);
  }
  if(temp_val_r != digitalRead(10)){
    Rindex++;
    temp_val_r = digitalRead(10);
  }
}
servoLeft.writeMicroseconds(1500);
servoRight.writeMicroseconds(1500);
delay(500);
Lindex = 0;
Rindex = 0;
```

```
temp_val_l = digitalRead(11);
temp_val_r = digitalRead(10);
forward();
while(1){
  flag_coll_des = collision_destination();
  if(flag_coll_des != 1){
    Serial.println("flag_coll_Dest");
    Serial.println(flag_coll_des);
    break;
  }
  if(Rindex > 10||Lindex > 10){//Forward 5cm
    break;
  }
  if(temp_val_l != digitalRead(11)){
    Lindex++;
    temp_val_l = digitalRead(11);
  }
  if(temp_val_r != digitalRead(10)){
    Rindex++;
    temp_val_r = digitalRead(10);
  }
}
servoLeft.writeMicroseconds(1500);
servoRight.writeMicroseconds(1500);
number = 0;
```

```
  }
}
void forward(){//Forward
  servoLeft.writeMicroseconds(1700);
  servoRight.writeMicroseconds(1300);
}
void turnRight(){//Right
  servoLeft.writeMicroseconds(1700);
  servoRight.writeMicroseconds(1700);
  //delay(time);
}
void turnLeft(){//Left
  servoLeft.writeMicroseconds(1300);
  servoRight.writeMicroseconds(1300);
  //delay(time);
}
void disableServos(){//disable servos
  servoLeft.detach();
  servoRight.detach();
}
// callback for received data
void receiveData(int byteCount){
  while(Wire.available()) {
    Rindex = 0;
    Lindex = 0;
```

```
    number = Wire.read();
    Serial.print("Data␣received:␣");
    Serial.println(number);
  }
}
// callback for sending data
void sendData(){
  Wire.write(flag_coll_des);
}
int collision_destination(){
  int col_des=0;
  byte wLeft = digitalRead(5);
  byte wRight = digitalRead(7);
  line_follower = analogRead(0);
  if(line_follower > 800){//Arrive at the destination
    Serial.println("Arrive␣at␣Destination");
    col_des = 2;
  }else if((wLeft == 0)&&(wRight==0)){//Both whisker detect
    Serial.println("Both␣Collision");
    col_des=5;
  }else if(wRight== 0){//Right whisker detects
    Serial.println("Right␣Collision");
    col_des=4;
  }else if(wLeft == 0){//Left whisker detects
    Serial.println("Left␣Collision");
```

```
        col_des=3;
    }else{
        col_des = 1;
    }
    return col_des;
}
```

### A.3 Hardware Python Code

```
#Real Model-free Reinforcement Learning
#Energy Efficiency Q-Learning

import warnings
warnings.filterwarnings('ignore')
import smbus
import learning, pygame, time, sys, Sprite_SJ
import math
import random
import cPickle
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
bus = smbus.SMBus(1)
address_1 = 0x04#Slave
sys.setrecursionlimit(15000)


#Ultrasonic Sensors
```

```
TRIG_1 = 17
ECHO_1 = 18
TRIG_2 = 27
ECHO_2 = 22
TRIG_3 = 23
ECHO_3 = 24
TRIG_4 = 25
ECHO_4 = 5
TRIG_5 = 6
ECHO_5 = 12
GPIO.setup(TRIG_1,GPIO.OUT)
GPIO.setup(ECHO_1,GPIO.IN)
GPIO.setup(TRIG_2,GPIO.OUT)
GPIO.setup(ECHO_2,GPIO.IN)
GPIO.setup(TRIG_3,GPIO.OUT)
GPIO.setup(ECHO_3,GPIO.IN)
GPIO.setup(TRIG_4,GPIO.OUT)
GPIO.setup(ECHO_4,GPIO.IN)
GPIO.setup(TRIG_5,GPIO.OUT)
GPIO.setup(ECHO_5,GPIO.IN)
GPIO.output(TRIG_1,False)
print "Waiting␣For␣Sensor␣to␣Settle"
time.sleep(0.5)
GPIO.output(TRIG_2,False)
time.sleep(0.5)
```

```
GPIO.output(TRIG_3,False)
time.sleep(0.5)
GPIO.output(TRIG_4,False)
time.sleep(0.5)
GPIO.output(TRIG_5,False)
time.sleep(0.5)

#Get Q table with reward from Software Simulation
with open\
('Updated_Collect_Data_NewMap_5_DF9_RE10_04.txt') \
as savefile:
    myBigList = cPickle.load(savefile)
carAgent = myBigList

#Get Q table from other simulation

Q=[]
ACTIONLIST = [1,2,3]
lastState = ()
lastAction = 0

# DISPLAY SETTING IN PYGAME

# Q–LEARNING
class static:
```

```
    FLAG = 1
    REWARD = 0

def writeNumber_1(value):
    bus.write_byte(address_1, value)
    # bus.write_byte_data(address, 0, value)
    return -1

def readNumber_1():
    number = bus.read_byte(address_1)
    # number = bus.read_byte_data(address, 1)
    return number

#Measure Distance from Ultrasonic sensor
def distance_ultra_betw_agent_obs(trig_pin,echo_pin):
    distance = 0
    GPIO.output(trig_pin,False)
    time.sleep(0.5)
    GPIO.output(trig_pin,True)
    time.sleep(0.00001)
    GPIO.output(trig_pin,False)

    while GPIO.input(echo_pin)==0:
        pulse_start=time.time()
```

```
    while GPIO.input(echo_pin)==1:
        pulse_end=time.time()

    pulse_duration = pulse_end - pulse_start
    distance = pulse_duration * 17150
    distance = round(distance , 2)
    return round(distance/2.54, 2)#inch#

def game_loop():
    temp = []
    state = temp
    #Ultraspmoc sensors
    s = [0,0,0]
    s[0] = distance_ultra_betw_agent_obs(TRIG_1,ECHO_1)
    print "Ultrasonic 1 : " + str(s[0])
    time.sleep(0.5)
    #s[1] = distance_ultra_betw_agent_obs(TRIG_2,ECHO_2)
    #print "Ultrasonic 2 : " + str(s[1])
    #time.sleep(1)
    s[1] = distance_ultra_betw_agent_obs(TRIG_3,ECHO_3)
    print "Ultrasonic 3 : " + str(s[1])
    #time.sleep(1)
    #s[3] = distance_ultra_betw_agent_obs(TRIG_4,ECHO_4)
    #print "Ultrasonic 4 : " + str(s[3])
    time.sleep(0.5)
```

```
s[2] = distance_ultra_betw_agent_obs(TRIG_5,ECHO_5)
print "Ultrasonic 5 : " + str(s[2])
time.sleep(0.5)
# STATE DEFINITION
list_temp = [0,0,0]
for n in range(3):
    list_temp[n] = s[n]

    if(0 <= s[n] < 10):
        list_temp[n] = 0
    elif(10 <= s[n] < 22):
        list_temp[n] = 1
    elif(22 <=s[n] < 34):
        list_temp[n] = 2
    elif(34<=s[n]<70):
        list_temp[n] = 3
    else:
        list_temp[n]=0
temp = list_temp
state = temp
state = tuple(state)
static.FLAG = carAgent.chooseAction(state)
print "state : " + str(state)
print "action : " + str(static.FLAG)
#Send to Arduino Uno
```

```
        writeNumber_1(static.FLAG)
        #Read value from Arduino Uno
        number = readNumber_1()
        while 1:
            number = readNumber_1()
            if number != 0:
                break

        print "number : " + str(number)
        if(number==1):
            print "Nothing happens"
            game_loop()
        elif(number == 2):
            print 'arrive'
            quit()
        else:
            print 'crash'
            quit()

while 1:
    game_loop()
    pygame.quit()
    quit()
```